n° d'ordre : Année 2004

THESE

présentée
pour obtenir

le titre de Docteur de l'Institut National Polytechnique de Toulouse

ECOLE DOCTORALE SYSTEMES

Spécialité : Informatique Industrielle

par

El-Djillali TALBI

Sélection et réglage de paramètres pour l'optimisation de logiciels d'ordonnancement industriel

Soutenance prévue le 12 novembre 2004 devant le jury composé de :

Colette MERCÉ	Professeur à l'INSA de Toulouse	Présidente
Patrick SIARRY	Professeur à l'Université de Paris XII Val-de-Marne	Rapporteur
Michel GOURGAND	Professeur à l'UBP de Clermont-Ferrand	Rapporteur
Bernard GRABOT	Professeur à l'ENI de Tarbes	Directeur de thèse
Laurent GENESTE	Professeur à l'ENI de Tarbes	co-Directeur de thèse
Romuald PREVITALI	Responsable R&D, Finmatica France	Examinateur

Thèse réalisée en **C**onvention **I**ndustrielle de **F**ormation par la **Re**cherche, Département R&D, Finmatica France SA, 150 grande rue de St Clair – Le Sextant, 69731 CALUIRE ET CUIRE CEDEX / Équipe Production Automatisée - Laboratoire Génie de la Production, École Nationale d'Ingénieurs de Tarbes, 47 Avenue d'Azereix, BP 1629, 65016 TARBES CEDEX

Remerciements

Mes remerciements les plus vifs et chaleureux, empreints d'une reconnaissance ineffable, vont à mon directeur de thèse Bernard Grabot et mon co-directeur Laurent Geneste tous deux professeurs à l'École Nationale d'Ingénieurs de Tarbes, pour leur aide, leur confiance, leurs orientations judicieuses et leur disponibilité.

Je tiens à remercier tout particulièrement monsieur Pascal Hostachy, responsable de l'équipe de développement de Finmatica France, qui a suivi avec rigueur mon travail durant ces quatre années de thèse, qu'il en soit humblement remercié.

Ce travail n'aurait pas vu le jour sans l'appui du responsable R&D de Finmatica France, monsieur Romuald Previtali qui m'a guidé dans la partie initiale de cette recherche et qui a accepté d'être membre du jury de soutenance en tant qu'examinateur. Je l'en remercie sincèrement.

Je remercie tout particulièrement monsieur le Professeur Patrick Siarry, de l'Université de Paris XII-Val de Marne, d'avoir accepté la lourde tâche de rapporteur. Je le remercie pour l'analyse minutieuse qu'il a menée sur le manuscrit, pour la lecture attentive qu'il en a fait et qui a contribué à son enrichissement et sa mise en forme.

J'aimerai exprimer ma gratitude à monsieur le Professeur Michel Gourgand, de l'Université Blaise Pascal de Clermont-Ferrand, pour avoir accepté d'être rapporteur.

Je remercie le Professeur Colette Mercé, de l'Institut National de Sciences Appliquées de Toulouse, d'avoir accepté de présider le jury de soutenance.

Je tiens à remercier monsieur Marc Schonauer, directeur de recherche à l'INRIA, pour ses conseils concernant l'implémentation des algorithmes évolutionnaires.

Cette thèse ne se serait pas passer dans d'aussi bonnes conditions sans tous les membres du service R&D de Finmatica France, qu'ils trouvent ici l'expression de ma gratitude.

Table des matières

Liste des publications

1. Journaux internationaux avec comité de sélection

1.1. Talbi, E. D., Geneste, L., Grabot, B., Previtali, R., and Hostachi, P. (2004). Application of optimization techniques to parameter set-up of industrial scheduling software. *Computers In Industry* 55, pp. 105-124.

2. Conférences internationales avec comité de sélection et actes

2.1. Talbi, D., Geneste, L., Grabot, B., Prévitali, R., and Hostachy, P. (2002). Optimal Set-up of an Industrial Scheduling Software. *APMS 2002 Collaborative Systems for Production Management*, Eindhoven, Pays-Bas, September 8-13.

2.2. Talbi, E. D., Geneste, L., and Grabot, B. (2003). Meta-heuristics for optimal set-up of an industrial scheduling software. *CESA'2003, 9-11 juillet 2003*, Lille, France.

3. Conférences nationales avec comité de sélection et actes

3.4. Talbi, E. D., Grabot, B. et Geneste, L. (2004). Algorithmes évolutifs pour le paramétrage d'un logiciel d'ordonnancement. *MOSIM'04*, Nantes, France.

4. Journées de travail et autres présentations

4.1. Talbi, E. D. (2002). Aide au choix et au réglage de paramètres dans un logiciel d'ordonnancement. 3ème congrès des doctorants de l'Ecole Doctorale Systemes, 22 Mai, Toulouse.

4.2. Talbi, E. D., Geneste, L. et Grabot, B. (2003). Méta-heuristiques pour le paramétrage automatique d'un logiciel d'ordonnancement industriel. Journée Bermudes / Métaheuristiques, 7 février 2003, Lille.

4.3. Talbi, D., Geneste, L. et Grabot, B. (2003). Méta-heuristiques pour le paramétrage automatique d'un logiciel d'ordonnancement industriel. Les journées de travail du GPR, 20-21 mars, Saint-Etienne.

Avant propos

Cette thèse s'est déroulée dans le cadre d'une Convention Industrielle de Formation par la Recherche (CIFRE) entre l'École Nationale d'Ingénieurs de Tarbes (ENIT) et la société Ortems (à l'heure actuelle, Ortems fait partie du groupe Finmatica France, après son rachat par le groupe Finmatica en 2002) éditeur du logiciel d'ordonnancement industriel Ortems®. Le travail de recherche lié à la thèse a été réalisé au sein du service R&D de la société Ortems et du laboratoire de génie de la production de l'ENIT, sous la direction de monsieur Bernard Grabot et la co-direction de monsieur Laurent Geneste tous deux professeurs à l'ENIT.

Cette contribution s'insère dans le cadre des travaux de recherche sur le développement d'outils d'aide à la décision pour le contrôle des systèmes de production, menées par l'équipe de Production Automatisée du laboratoire de génie de la production de L'ENIT.

Nous proposons dans cette thèse une approche visant à améliorer les performances d'un logiciel d'ordonnancement en intervenant au niveau de son paramétrage. L'idée que nous développons ici est d'utiliser des métaheuristiques pour automatiser le réglage des paramètres intervenant en entrée du logiciel. Deux problèmes seront abordés : la sélection des paramètres pertinents à l'entrée du logiciel et le réglage de ces paramètres en fonction des exigences de l'utilisateur en terme de performances à atteindre par l'ordonnancement. Ces deux problèmes sont très liés et ne peuvent être résolus indépendamment l'un de l'autre. Les choix effectués au moment de la sélection de paramètres conditionnent fortement la qualité des réglages obtenus. Pour cette raison, nous proposons ici des méthodes permettant de résoudre les deux problèmes de façon

simultanée, en introduisant des stratégies de sélection de paramètres au sein des métaheuristiques. L'approche proposée est appliquée au logiciel d'ordonnancement Ortems® et validée sur plusieurs cas industriels.

Le chapitre 0 est une introduction générale à la problématique de paramétrage de logiciels d'ordonnancement, ainsi qu'une description du cas particulier du logiciel d'ordonnancement Ortems®.

Dans le chapitre 1, nous aborderons le problème général de l'optimisation en boîte noire de systèmes complexes : les aspects théoriques et les principales techniques et approches proposées dans la littérature, ainsi que leurs applications au cas particulier des systèmes de production.

Le chapitre 2 sera consacré à la description du cadre général de nos approches et des algorithmes utilisés.

Le chapitre 3 est composé de trois parties indépendantes. Dans la première seront exposés les aspects liés à l'implémentation (architecture utilisée), ainsi qu'une description fonctionnelle et technique du prototype *Ortems Optimizer*, dans lequel ont été implémentées les approches décrites au chapitre 2. Les deux dernières parties du chapitre seront consacrées au problème du paramétrage du logiciel d'ordonnancement industriel Ortems. Celui-ci sera décrit dans la deuxième partie du chapitre 3. Puis dans une troisième partie nous présenterons un ensemble d'expérimentations numériques effectuées sur plusieurs cas industriels, afin de valider nos approches, tant du point de vue pratique que théorique.

Dans la partie finale de la thèse, nous donnerons quelques conclusions liées à la mise en oeuvre des approches proposées, les différentes perspectives de recherche qui peuvent être envisagées dans le futur, suivies des trois annexes A, B et C. L'annexe C décrira une première expérience réussie de l'application des approches décrites dans cette thèse à un cas industriel réel.

Chapitre 0

Introduction générale

0.1. L'ordonnancement d'atelier

Résoudre un *problème d'ordonnancement* consiste à organiser dans le temps la réalisation de tâches, de façon à satisfaire un ou plusieurs objectifs, et en prenant en compte un certain nombre de contraintes temporelles (délais, contraintes d'enchaînement) et de contraintes portant sur la disponibilité des ressources requises. Les problèmes d'ordonnancement se rencontrent très souvent dans le milieu industriel, notamment dans l'optimisation de la gestion des systèmes de production.

Les approches traditionnelles pour résoudre ce problème consistent à appliquer des techniques d'optimisation à une formulation analytique. Cependant, ces modèles ont montré leurs limites [Buzacott et Yao, 1986] [Ben-Arieh, 1988] [Newman, 1988]. Même des formulations idéalistes et simplifiées peuvent s'avérer très difficiles à traiter. Pour trouver un optimum, le seul moyen est souvent de faire une recherche exhaustive sur l'ensemble des solutions réalisables. Le développement de la théorie de la complexité des algorithmes a permis de clarifier la difficulté du problème [Rinnooy Kan, 1976] [Garey et Johnson, 1979]. La plupart des problèmes d'ordonnancement sont NP-difficiles [Lawler et al., 1993]. Il s'ensuit que la plupart des problèmes de taille industrielle (plusieurs milliers de tâches, ressources complexes et contraintes spécifiques) sont impossibles à résoudre de manière exacte.

Les chercheurs se sont alors orientés vers l'utilisation de méthodes approchées appelées « heuristiques ». Contrairement à une méthode exacte, qui vise à l'obtention

d'une solution optimale, l'objectif d'une heuristique est de trouver une « bonne » solution en un temps raisonnable. Les heuristiques sont beaucoup plus adaptées au contexte industriel, où il ne s'agit pas tant de satisfaire complètement tel ou tel objectif que de fournir une solution d'ordonnancement réaliste, qui procure une satisfaction jugée acceptable d'un ensemble d'objectifs. Pour cette raison, les logiciels d'ordonnancement disponibles sur le marché font tous appel à des méthodes heuristiques, principalement basées sur trois types d'approches [Grabot et Geneste, 1994] :

- Placement d'ordres de fabrication : logiciels anciens comme *Saveplan* ou *Prodstar* (on veut obtenir vite une solution, sans objectif de performance marqué),
- Raisonnement à base de contraintes : *Ordo* (EBC Informatique)
- Simulation : *SipaPlus* (Grai Industries), *CadPlan* (Access Commerce), *Ortems* (Finmatica), *Préactor* (Cabinet Ouroumoff), *Io* (Cesium), *Véga* (Cybernetix)…

L'approche par simulation est de loin la plus répandue dans les logiciels industriels. Elle repose sur l'utilisation de règles de priorité sur les tâches, appelées aussi « règles d'ordonnancement » (par exemple : priorité aux tâches les plus courtes, aux tâches les plus en retard…). La littérature relative à ces règles est particulièrement riche (voir par exemple [Sabuncuoglu, 1998], pour un état de l'art).

On trouve dans [Wu, 1987] la classification suivante pour les règles d'ordonnancement apparues dans la littérature :

- *Règles simples :* basées sur des informations sur les tâches à ordonnancer. Des exemples sont la règle de la durée minimale (SPT), du délai minimum (EDD), de la marge minimale (MINSLACK), du premier arrivé premier servi (FIFO) ….
- *Combinaison de règles simples :* par exemple, « appliquer SPT tant que la longueur de la file d'attente ne dépasse pas 5, ensuite appliquer FIFO ». Ainsi, on évite que les tâches de longue durée ne restent dans la file d'attente trop longtemps.
- *Indices de priorité pondérés « Weight Priority Indexes » :* c'est une somme pondérée d'indices de priorité, chaque indice de priorité étant une fonction numérique quantifiant une information sur la tâche.

Si ces règles procurent en général des temps de réponse acceptables, il est difficile de choisir une règle en fonction de la satisfaction d'un ensemble d'objectifs donné. Celle-ci varie en effet en fonction du cas traité (nombre de machines, nombre de tâches dans les gammes, durées des tâches, …etc.). Plusieurs études ont été menées pour identifier la meilleure règle d'ordonnancement en fonction de la configuration de l'atelier et des conditions d'exploitation [Panwalker et Iskander, 1977] [Graves, 1981] [Blackstone et al., 1982] [Baker, 1984] [Rodammer et White, 1988] [Grabot et Geneste, 1994] [Pierreval et Mebarki, 1997]. De telles études se sont, à notre connaissance, cantonnées à des problèmes d'ordonnancement simples et n'ont pas été généralisées dans un contexte industriel.

Les logiciels d'ordonnancement industriels disponibles sur PC, qui sont apparus sur le marché dans les années 1985-1995, se sont d'abord principalement focalisés sur

deux aspects : exécution rapide et utilisation conviviale. L'aspect optimisation n'avait que peu ou pas d'importance. A l'heure actuelle, le contexte a changé : les planificateurs ne sont plus satisfaits par les solutions réalisables qui leur sont proposées et veulent optimiser l'ordonnancement par rapport à la satisfaction de certains objectifs, pour accroître la compétitivité de leur production [Grabot, 1998].

Comparées aux autres méthodes d'ordonnancement, celles utilisant les règles d'ordonnancement offrent l'avantage d'être très faciles à implémenter et d'avoir le temps de calcul le plus faible, ce qui les rend bien adaptées aux problèmes de grandes tailles. Cependant, ces méthodes présentent de nombreux inconvénients [Subramaniam et al., 2000] :

- Ces règles ne considèrent pas toutes les ressources disponibles à un instant donné. Généralement, une règle supplémentaire est ajoutée pour sélectionner une ressource avant d'appliquer la règle d'ordonnancement.
- Plusieurs études ont montré qu'une combinaison de règles simples peut donner de meilleurs résultats par rapport à l'utilisation d'une seule règle [Lawrence, 1984] [Grabot et Geneste, 1994] [Dorndorf et Pesch, 1995].
- Il n'existe aucune règle meilleure que toutes les autres dans toutes les situations. La littérature décrit des centaines de règles, et le choix d'une règle adaptée dépend de la nature du problème d'ordonnancement et de la mesure de performance considérée. De plus, l'environnement d'ordonnancement étant dynamique, la nature du problème d'ordonnancement change au cours du temps, et ces règles doivent donc aussi changer au cours du temps [Subramaniam, 1995].

Pour améliorer les résultats obtenus en utilisant des règles d'ordonnancement, certains auteurs se sont orientés vers l'utilisation de techniques appartenant au domaine de l'intelligence artificielle, en particulier les systèmes experts. Les règles d'ordonnancement ne sont plus fixées une fois pour toutes, mais suggérées par un système expert, en fonction du contexte courant de l'atelier. Le système expert utilise une base de connaissances composée de règles d'ordonnancement, obtenues à partir de l'expérience accumulée par certains individus clés [Toal et al., 1994]. Plusieurs projets ont été menés pour développer de tels systèmes : *ESPRIT 418* [Milin, 1987], *ESPRIT 809* [Van der Pluym, 1990], *ESPRIT 6805* [ESPRIT 6805, 1995], et plusieurs prototypes ont été développés : *OPAL/ORIGAN* [Bensana et al., 1988], *BOSS* [Hynynen, 1988], *OPIS* [Smith et al., 1990]. Jusqu'ici, aucun de ces systèmes n'a eu un vrai impact industriel [Grabot, 1998], pour deux raisons essentielles [Steffen, 1986] :

- la non généricité des connaissances acquises,
- l'énorme effort nécessaire pour générer une base de connaissances adaptée à un atelier donné.

Les métaheuristiques [Dréo et al., 2003], une autre famille d'algorithmes approchés, ont également suscité un grand nombre de travaux de recherche appliqués à l'ordonnancement au cours des deux dernières décennies. Le terme *métaheuristique* provient de la composition de deux mots grecs. « Heuristic » dérive du verbe « heuristikein », qui signifie « trouver », et le préfixe « méta » signifie « au dessus, dans un niveau supérieur ». Le principe général d'une métaheuristique est d'explorer l'espace

de recherche en combinant plusieurs heuristiques de base dans des structures de niveau plus élevé.

Nous verrons par la suite comment améliorer les performances d'un logiciel d'ordonnancement industriel en intervenant au niveau de son paramétrage. Une approche basée sur l'utilisation de métaheuristiques est proposée et un environnement d'optimisation appelé *Ortems Optimizer* est conçu et appliqué au logiciel d'ordonnancement *Ortems*.

0.2. Problématique

0.2.1. Problématique du paramétrage du logiciel *Ortems*

L'algorithme d'ordonnancement d'*Ortems* est paramétrable grâce à une configuration de règles de priorité (voir la description de l'algorithme au chapitre 3). Définir une « bonne » configuration de règles est une tâche très importante, à cause de sa grande influence sur les performances de l'ordonnancement, mais très difficile.

A l'heure actuelle, les configurations de règles sont spécifiées par un consultant à l'installation du logiciel, de façon manuelle par les consultants d'*Ortems* en se basant sur leur expérience. Plusieurs jours sont alors nécessaires pour aboutir à une configuration acceptable par le client. Cependant, cette approche manuelle présente plusieurs points faibles :

- plusieurs règles ne sont jamais utilisées, en raison du manque de connaissances des consultants de l'effet de certaines règles et du nombre important de règles disponibles (environ 40 règles d'ordonnancement et 20 règles de placement). De plus, *Ortems* offre la possibilité à l'utilisateur de construire des règles spécifiques, au moyen de sa plate-forme de développement *user exit*,
- il est difficile de satisfaire les objectifs du client, souvent contradictoires (minimiser les coûts de production, éviter les retards de livraison, maximiser l'utilisation des équipements …),
- la configuration proposée par le consultant, même si elle est bien adaptée à l'environnement de production initial, peut l'être beaucoup moins après un certain temps [Chandra et Talavage, 1991]. Ceci est dû aux changements dans les données de l'environnement de production considéré : arrivée de nouveaux ordres de fabrication, changements dans les délais client, pannes machines …. Ceci pose le problème de la *robustesse temporelle* de la configuration de règles,
- le manque d'expérience du client fait qu'il est généralement incapable de reconfigurer les règles pour une meilleure performance de l'ordonnancement,
- le problème de pertinence de l'expérience acquise en ordonnancement,
- la plupart du temps, le consultant dispose de très peu de temps pour proposer une configuration de règles, alors qu'un réglage manuel des paramètres est une approche par essai-erreur très coûteuse en temps, d'autant plus que, dans

certaines situations, le temps requis pour calculer un ordonnancement peut être très élevé (plusieurs dizaines de minutes).

Pour faire face à toutes ces difficultés, la société *Ortems* a initialisé un programme de recherche, objet de cette thèse, dont le but est d'automatiser la tâche de paramétrage des règles de priorité utilisées par un logiciel d'ordonnancement (quelconque), et de valider l'approche proposée sur le logiciel *Ortems*.

L'approche de « Quasi Ordre Lexicographique » (QOL) utilisée pour combiner les règles a pour cela été remplacée par une approche de type « Indices de Priorité Pondérés » (IPP). Ce choix de remplacer l'approche QOL par IPP a été fait essentiellement pour balayer un spectre plus large de solutions. En effet, dans une approche de type QOL :

- les 3 ou 4 premières règles ont une grande influence sur les résultats de l'ordonnancement, les autres règles n'ayant que très peu ou pas d'influence,
- les situations où plusieurs règles ont la même importance ne sont pas prises en compte.

Étant données les règles élémentaires R_1, R_2, ..., R_k et un ensemble C de tâches ou de machines, le choix d'un élément de C est effectué en résolvant le problème d'optimisation :

$$\underset{x \in C}{Min} \sum_{i=1}^{k} w_i \, \bar{h}_{R_i}(x),$$

où w_i, $(1 \le i \le k)$ est le poids ($0 \le w_i \le 1$ et $\sum_{i=1}^{k} w_i = 1$) associé à la règle R_i,

et $\bar{h}_R$ est la valeur normalisée de la fonction numérique h_R associée à la règle R (voir la description de la méthode d'ordonnancement d'*Ortems* en 3.3 (chapitre 3)).

Cette normalisation est importante, car elle permet de ramener à une même échelle les fonctions numériques associées aux différentes règles. Plusieurs procédures de normalisation existent dans la littérature, nous renvoyons à [Pomerol et Barba-Romero, 1993] pour plus de détails. La méthode de normalisation que nous avons adoptée est la suivante :

$\bar{h}_R = \dfrac{h_R - m_{R,C}}{M_{R,C} - m_{R,C}}$, si h_R est à minimiser,

et $\bar{h}_R = \dfrac{M_{R,C} - h_R}{M_{R,C} - m_{R,C}}$, si h_R est à maximiser,

où $M_{R,C}$ est le maximum de h_R sur C, et $m_{R,C}$ le minimum de h_R sur C.

0.2.2. Problématique générale

La problématique présentée dans la section précédente n'est pas spécifique à *Ortems* mais se retrouve dans la plupart des logiciels existant sur le marché [Lereno et al., 2001].

Les points suivants caractérisent en effet la plupart des logiciels d'ordonnancement commercialisés :

- *Un grand nombre de paramètres difficiles à régler :* en raison de la diversité des problèmes d'ordonnancement rencontrés en pratique, le nombre de paramètres intervenant dans les logiciels d'ordonnancement est de plus en plus élevé. L'éditeur d'un logiciel d'ordonnancement ne peut pas redévelopper des parties de son moteur d'ordonnancement à chaque fois qu'il se trouve confronté à une nouvelle situation pratique. Cette approche serait trop coûteuse et impossible à maintenir. Plusieurs chercheurs ont proposé de recourir à des logiciels configurables, qui peuvent s'adapter à des situations très différentes, souvent inattendues, rencontrées en pratique [Fourer et al., 1993] [Davis et Fox, 1994] [McIlhagga, 1997] [Smith et Becker, 1997] [Montana, 2001]. Cette approche consiste à mettre en place un certain nombre de paramètres qui pourront être réglés en fonction du problème à résoudre, offrant ainsi des degrés de liberté au logiciel pour s'adapter à chaque nouvelle situation. D'autre part, même si aucun client n'a besoin de toutes les fonctionnalités d'un logiciel aussi complexe que ceux existants sur le marché, et même s'il est admis qu'un logiciel spécialement développé pour résoudre un problème spécifique donne des résultats meilleurs, les sociétés préfèrent s'orienter vers des logiciels standards, en raison du coût élevé et de la faible possibilité d'évolution d'un développement spécifique.
- *Des résultats d'ordonnancement très sensibles aux valeurs des paramètres :* l'effet d'un paramètre sur la qualité d'un ordonnancement dépend fortement des caractéristiques du cas industriel traité (nombre de ressources, charge moyenne, dispersion des temps de traitement…) [Sim et al., 1994] [Pierreval et Mebarki, 1997].
- *Les valeurs des paramètres sont en général fixées une fois pour toutes :* c'est au cours de l'installation du logiciel que les valeurs des paramètres sont ajustées. Elles sont rarement remises en cause, car les utilisateurs possèdent en général très peu d'expérience pour fixer les valeurs de ces paramètres. En pratique, les caractéristiques d'un atelier varient toutefois dans le temps : les capacités et la disponibilité des ressources, ainsi que la liste des tâches à ordonnancer sont en évolution permanente. De plus, les objectifs poursuivis sont souvent implicites et l'utilisateur a beaucoup de difficulté à les formaliser. C'est par l'utilisation du logiciel d'ordonnancement que le gestionnaire d'atelier prend de plus en plus conscience de ses besoins. En outre, l'importance relative des différents objectifs peut varier dans le temps [Pinedo, 1995]. Par conséquent, il est important de permettre au client d'adapter la méthode d'ordonnancement aux changements survenus dans l'environnement et les objectifs de production.
- *Les logiciels ne prennent pas en compte les préférences du planificateur :* parmi les logiciels commercialisés, rares sont ceux qui offrent la possibilité au planificateur de définir plusieurs objectifs, alors que les planificateurs sont souvent confrontés à des objectifs multiples et contradictoires : respecter les dates de livraison, réduire les coûts de production, ….

L'idée que nous proposons ici est d'automatiser le réglage des paramètres intervenant en entrée des logiciels d'ordonnancement par rapport à un problème donné. L'objectif sera de sélectionner les paramètres qui ont une influence sur la qualité de

l'ordonnancement, et d'ajuster leurs valeurs de façon à optimiser la qualité de l'ordonnancement.

La sélection de paramètres est intéressante pour plusieurs raisons :

- *Éviter l'explosion combinatoire de l'espace de recherche :* étant donné le nombre important de paramètres utilisés par un logiciel d'ordonnancement, on peut rapidement être confronté au problème crucial de l'explosion combinatoire de l'espace de recherche, aussi connu sous le nom de « curse of dimensionality » [Bellman, 1961] : la difficulté de l'optimisation s'accroît de façon exponentielle en fonction de la dimension de l'espace de recherche.
- *Réduire le temps d'ordonnancement :* si l'on considère le cas où les paramètres à spécifier sont ceux associés à des règles composées, comme c'est le cas pour *Ortems*, choisir un grand nombre de paramètres revient à utiliser un grand nombre de règles d'ordonnancement, ce qui peut rendre l'ordonnancement très coûteux en temps de calcul.
- *Améliorer l'interprétation des résultats :* souvent, les planificateurs ne veulent pas seulement optimiser leur ordonnancement, mais aussi expliquer le résultat obtenu et analyser les paramètres qui ont une influence sur ce résultat. Obtenir une solution en ajustant un petit nombre de paramètres permet d'expliquer comment cette solution a été obtenue, et donc de mieux comprendre le fonctionnement du système de production.

0.3. Notre approche

L'objectif de notre recherche est double :

- d'une part la conception et le développement d'un environnement d'aide à la décision permettant de :

  - *sélectionner* les paramètres pertinents à l'entrée d'un logiciel d'ordonnancement donné,

  - *régler* les valeurs de ces paramètres en fonction des exigences de l'utilisateur, en terme de performances à atteindre par l'ordonnancement,

- d'autre part, la réalisation de tests pratiques sur des cas d'application réels, en utilisant le logiciel d'ordonnancement *Ortems*.

Cet environnement doit obéir principalement à deux exigences : généralité (l'environnement doit être indépendant du logiciel considéré) et approche multicritère (donner à l'utilisateur la possibilité de définir ses préférences).

Notre approche pour résoudre les deux problèmes de sélection et de réglage de paramètres est fondée sur plusieurs concepts :

- *Optimisation en boîte noire :* le logiciel d'ordonnancement est considéré comme une boîte noire, avec des paramètres en entrée et des indicateurs de performance en sortie. L'objectif est de spécifier les valeurs des paramètres en entrée de façon à optimiser les indicateurs en sortie.

- *Utilisation de métaheuristiques,* qui sont des méthodes d'optimisation approchée destinées à la résolution des problèmes difficiles de l'optimisation combinatoire.
- *Optimisation et sélection, simultanées :* les deux problématiques de sélection et de réglage de paramètres sont interdépendantes. D'un côté, la qualité de l'optimisation est conditionnée par les paramètres sélectionnés, et d'un autre côté un groupe de paramètres sélectionnés ne peut être évalué qu'en ajustant leurs valeurs. Une approche qui associe l'optimisation et la sélection nous paraît donc adaptée.
- *Optimisation multicritère :* l'optimisation en présence de plusieurs objectifs pose un certain nombre de difficultés : définition des objectifs, difficultés théoriques liées à l'inexistence d'une solution optimale et choix, parmi la multitude d'approches multicritères décrites dans la littérature.

Comme nous le verrons dans le premier chapitre de ce document, plusieurs approches ont été proposées dans la littérature pour résoudre les deux problématiques de sélection et réglage de paramètres intervenant en ordonnancement. Les principaux points d'originalité de l'approche que nous proposons sont :

- Optimisation et sélection simultanées (basées sur l'introduction de techniques de sélection de variables au sein de métaheuristiques),
- Conception d'un environnement d'optimisation *générique*, *multicritère* et *interactif*,
- Développement d'un produit *Ortems Optimizer* commercialisable et directement utilisable par le planificateur,
- Validation de l'approche sur des problèmes réels de taille industrielle.

Chapitre 1

Techniques d'optimisation pour les problèmes de type « boîte noire déterministe »

1.1. Introduction

De manière générale, un algorithme spécifiquement conçu pour résoudre un problème particulier est plus performant qu'un algorithme générique. Cependant, dans certaines situations, seul un algorithme générique peut être appliqué. C'est le cas par exemple s'il n'y a pas assez de ressources (temps, ressources monétaires, et/ou connaissances) pour concevoir un algorithme spécifique [Wegener, 2000]. On se retrouve aussi dans cette situation lorsque l'on veut optimiser un système complexe dont les performances ne peuvent être évaluées que par simulation. Ce dernier cas est très fréquent en pratique. De nombreux systèmes, dans des domaines comme la production, la logistique ou la finance, sont en effet trop complexes pour être modélisés avec une formulation analytique.

Dans ce cas, on fait souvent appel à des outils de simulation permettant d'évaluer les performances de ces systèmes en fonction de variables de décision. La fonction objectif à optimiser est alors inconnue et sa valeur en un point de l'espace de recherche ne peut être obtenue qu'après une simulation (ou expérimentation numérique). C'est à cette catégorie de fonctions, appelées fonctions de type *boîte noire*, que nous nous intéresserons tout au long de ce chapitre.

Une fonction f à valeurs réelles est dite de type *boîte noire* si elle est définie dans un domaine X de $\mathbb{R}^n$ et vérifie les trois propriétés suivantes :

- le domaine X est connu,

- il est possible de connaître la valeur de *f* en tout point de X, grâce à une expérimentation numérique (ou simulation),
- cette dernière information est la seule disponible sur la fonction *f*.

Le problème qui consiste à optimiser de telles fonctions est dit *problème d'optimisation en boîte noire* (voir par exemple [Kargupta, 1995] pour une définition de ce type de problèmes).

Selon la nature des sorties du système à optimiser, on peut distinguer deux catégories de problèmes d'optimisation en boîte noire :

- *problèmes déterministes :* c'est le cas où les sorties du système sont reliées aux entrées par une fonction déterministe, c'est-à-dire n'utilisant aucun concept stochastique. Il s'ensuit qu'en utilisant les mêmes entrées, on obtient toujours les mêmes sorties.
- *Problèmes non déterministes :* certaines sorties du système sont liées aux entrées par une fonction aléatoire. Dans ce cas, les sorties peuvent varier si elles sont réévaluées en utilisant les mêmes entrées.

Nous nous intéresserons ici plus particulièrement au cas déterministe. Pour une revue des techniques d'optimisation destinées au cas non déterministe, plus connues sous le nom de *méthodes d'optimisation par simulation*, voir [Carson et Maria, 1997] [Ólafsson et Kim, 2002] [April et al., 2003].

Selon la nature de la solution recherchée pour un problème d'optimisation, on peut distinguer deux types de problèmes : l'optimisation locale[1] et l'optimisation globale. L'optimisation locale consiste à rechercher la meilleure solution localement, c'est-à-dire dans une région restreinte de l'espace de recherche. Dans le cas de l'optimisation globale en revanche, la solution recherchée est la meilleure sur tout l'espace de recherche. Ces deux types de problèmes ne s'excluent pas mutuellement. Afin d'améliorer les performances d'une recherche globale, de nombreux auteurs proposent d'utiliser une recherche globale pour bien explorer l'espace de recherche, conjointement avec une recherche locale pour mieux exploiter les zones prometteuses (voir par exemple les algorithmes mémétiques décrits en 1.2.1.1.2.4).

La difficulté des problèmes d'optimisation globale en boîte noire est bien connue. A moins de supposer certaines conditions sur la fonction objectif, toute méthode conçue pour résoudre un problème d'optimisation globale nécessite un nombre infini d'évaluations de la fonction objectif [Dixon, 1978] [Rinnooy Kan et Timmer, 1989]. En d'autres termes, sans aucune hypothèse sur la structure de la fonction à optimiser, aucun algorithme ne peut garantir de trouver un optimum global en un nombre fini d'itérations.

Le premier théorème de NFL (*No Free Lunch*) [Wolpert et Macready, 1995, 1997] montre que tous les algorithmes ont la même performance en moyenne lorsqu'ils sont évalués sur tous les problèmes d'optimisation en boîte noire possibles. Une conséquence importante de ce théorème est que, sans prise en compte d'informations

[1] On parle aussi d'algorithmes de recherche locale avec un sens différent : ce sont les algorithmes qui fonctionnent par voisins successifs. Ainsi, le recuit simulé est à la fois « local » (par son mécanisme de fonctionnement) et « global » (par sa finalité d'optimum global).

spécifiques à un problème donné, aucun algorithme n'est meilleur en moyenne qu'un échantillonnage aléatoire.

Les théorèmes de NFL ont été critiqués comme trop généraux pour être applicables, car en pratique nous ne nous intéressons pas à l'ensemble de tous les problèmes possibles, mais à un groupe beaucoup plus restreint de problèmes. C'est ce qui explique que les algorithmes d'optimisation en boîte noire donnent la plupart du temps de bons résultats (s'ils sont comparés à une recherche aléatoire), sur la plupart des problèmes qui nous intéressent. Ce dernier constat est appuyé par quelques résultats récents, qui montrent qu'il est peu vraisemblable qu'une fonction qui vérifie les théorèmes de NFL soit issue d'un problème pratique [Whitley, 2000] [Igel et Toussaint, 2003].

A cause de son intérêt pratique, l'optimisation en boîte noire a fait l'objet d'un nombre considérable de travaux de recherche au cours des deux dernières décennies. Cependant, une étude relativement récente de six logiciels d'optimisation globale en boîte noire [Mongeau et al., 2000] a abouti à la conclusion décevante que le temps requis par ces logiciels est trop élevé pour qu'ils soient utilisables dans la plupart des cas industriels.

1.2. Techniques d'optimisation en boîte noire

Nous proposons de classifier les techniques utilisées pour la résolution des problèmes déterministes de type boîte noire en trois classes (voir fig. 1.1) :

- *les approches basées sur l'évaluation exacte* de la fonction objectif,
- *les approches basées sur l'évaluation approchée*, appelées aussi approches par la *méta-modélisation*,
- *les autres approches*, qui peuvent combiner des méthodes appartenant aux deux classes précédentes.

Nous présenterons ici les principales méthodes appartenant aux deux premières classes (approches basées sur l'évaluation exacte et approches basées sur la méta-modélisation). Plusieurs autres approches ont été proposées, permettant de combiner des méthodes issues de ces deux classes [April et al., 2003] [Jin, 2003].

1.2.1. Approches basées sur l'évaluation exacte

Les *approches basées sur l'évaluation exacte* reposent sur l'évaluation exacte de la fonction objectif pour effectuer leur recherche dans l'espace des solutions. Les principales méthodes appartenant à cette catégorie sont les métaheuristiques, les méthodes de recherche directe et les méthodes statistiques.

1.2.1.1. Métaheuristiques

Selon la nature du processus de recherche mis en oeuvre, on peut classer les méthodes utilisées pour l'optimisation globale en deux catégories : les *méthodes déterministes* et les *méthodes stochastiques* [Rinnooy Kan et Timmer, 1989] [Dixon et Szegö, 1978]. Les méthodes déterministes requièrent des hypothèses restrictives sur la

fonction à optimiser telles que : la continuité, la dérivabilité ou les conditions de Lipchitz. De plus, ces méthodes ne sont applicables que pour des problèmes de faible dimension. Pour les autres problèmes, seules les méthodes stochastiques peuvent être utilisées [Archetti et Schoen, 1984].

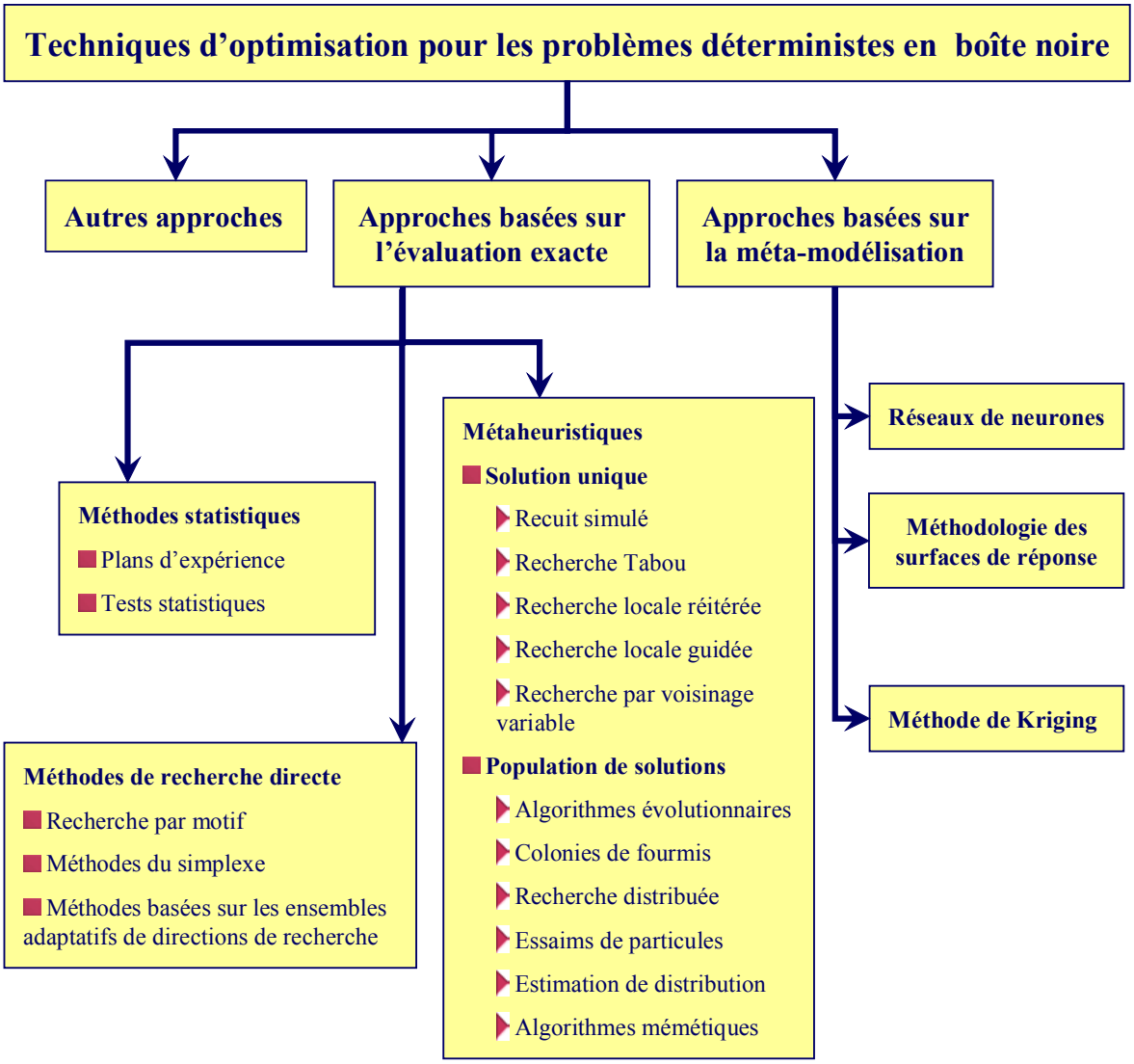


Figure 1.1 - *Classification des techniques d'optimisation pour les problèmes déterministes en boîte noire.*

Les méthodes stochastiques sont basées sur une approche en partie ou entièrement guidée par un processus stochastique. Contrairement aux méthodes déterministes, leur convergence n'est pas garantie, ou, dans le meilleur des cas, elle est garantie de manière asymptotique (c'est-à-dire pour un nombre infini d'itérations). La méthode stochastique la plus simple est l'échantillonnage aléatoire (ou méthode de Monte-Carlo) qui consiste à évaluer des points engendrés de façon aléatoire et à conserver le meilleur. Son avantage est d'être simple et très facile à implémenter, mais elle présente un sérieux

inconvénient : elle n'est pas efficace, et exige souvent un nombre élevé d'itérations pour donner une solution acceptable.

Parmi les différentes méthodes stochastiques d'optimisation globale, nous allons nous intéresser aux *heuristiques* (ou *méthodes approchées*). Contrairement aux méthodes dites *exactes,* les heuristiques ne procurent pas forcément une solution optimale, mais seulement une *bonne* solution en fonction du temps disponible. Une heuristique peut être conçue pour résoudre un type de problème donné, ou bien comme une méthode générale applicable à divers problèmes d'optimisation. Dans le second cas, elle est désignée sous le terme de *métaheuristique.* Cette définition d'une métaheuristique est celle adoptée par le « Metaheuristics Network » (www.metaheuristics.org): « A metaheuristic is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems ». Cependant, étant donnée la grande diversité de techniques et concepts utilisés par les métaheuristiques, il n'existe jusqu'ici aucune définition communément acceptée. Pour une liste de définitions proposées dans la littérature, ainsi qu'un état de l'art récent, voir [Blum et Roli, 2001] [Dréo et al., 2003].

Plusieurs classifications des métaheuristiques ont été proposées ; la plupart distinguent globalement deux catégories : les méthodes à base de solution courante unique, qui travaillent sur un seul point de l'espace de recherche à un instant donné, comme les méthodes de recuit simulé, recherche tabou et recherche locale réitérée, et les méthodes à base de population, qui travaillent sur un ensemble de points de l'espace de recherche, comme les algorithmes évolutionnaires, les colonies de fourmis et la recherche par dispersion.

1.2.1.1.1. Méthodes à base de solution courante unique

Dans les problèmes d'optimisation où l'on cherche à optimiser une fonction objectif sur un espace de décision donné, une petite perturbation sur un point de cet espace induit souvent une petite variation des valeurs de la fonction objectif en ce point. On en déduit que les bonnes solutions ont tendance à se trouver à proximité d'autres bonnes solutions, les mauvaises étant proches d'autres mauvaises solutions. D'où l'idée qu'une bonne stratégie consisterait à se déplacer à travers l'espace de recherche en effectuant de petits pas (petits changements sur le point courant) dans des directions qui améliorent la fonction objectif. Cette idée est à la base d'une grande famille d'algorithmes appelée *méthodes de recherche locale*, dont la version la plus simple est la *méthode de descente*[2]. Celle-ci procède de manière itérative, en choisissant à chaque itération un point dans le *voisinage*[3] de la solution courante. S'il est meilleur, il devient la nouvelle solution courante, sinon un autre point est choisi, et ainsi de suite. Une excellente introduction à cette méthode peut être trouvé dans [Papadimitriou et Steiglitz, 1982].

Plusieurs versions de la méthode de descente ont été proposées :

- *descente aléatoire :* la nouvelle solution est choisie aléatoirement,

---

[2] Certains auteurs utilisent aussi les termes *amélioration itérative* ou *recherche locale simple* pour dénommer cette méthode.

[3] Le voisinage d'un point x de l'espace de recherche peut être défini comme un ensemble de points de l'espace considérés comme « proches » de x.

- *descente déterministe :* le meilleur voisin est choisi,
- *descente vers le premier meilleur :* le premier voisin meilleur que la solution courante est choisi.

Le principal inconvénient de la méthode de descente est qu'elle reste piégée dans le premier optimum local rencontré. Pour faire face à ce problème, plusieurs méthodes de recherche locale ont été développées pour tenter, au moyen de différentes stratégies, d'éviter de rester bloqué dans un optimum local.

1.2.1.1.1.1. La recherche locale répétée

La méthode la plus simple permettant d'améliorer une solution obtenue par une méthode de descente consiste à redémarrer la descente plusieurs fois, en utilisant des solutions initiales différentes. Cette méthode, appelée *recherche locale répétée*, a produit des solutions excellentes sur certains problèmes pratiques [Beveridge et al., 1996], et dans certains cas elle a réussi à être plus performante que certaines métaheuristiques plus sophistiquées comme le recuit simulé [Johnson et McGeoch, 1997], les algorithmes génétiques et la programmation génétique [Juels et Wattenberg, 1996].

1.2.1.1.1.2. Recuit simulé

La méthode du recuit simulé tire son origine de la physique statistique. Son principe de fonctionnement repose sur une imitation du phénomène de recuit en science des matériaux. Le recuit est le processus physique qui consiste à porter un matériau à température élevée, puis à le refroidir d'une manière contrôlée jusqu'à l'obtention d'une structure régulière, comme dans les cristaux et l'acier. Durant ce processus, l'énergie du matériau est minimisée, en évitant de rester piégé dans certains états correspondant à des optimums locaux [Aarts et Korst, 1989].

En faisant une analogie avec le processus physique décrit ci-dessus, trois chercheurs de la société IBM (S. Kirkpatrick, C. D. Gelatt et M. P. Vecchi) ont proposé une méthode permettant d'apporter une solution aux problèmes difficiles de l'optimisation combinatoire : le recuit simulé [Kirkpatrick et al., 1983][4]. Comme la méthode de descente, cette méthode est procédure itérative qui consiste à engendrer, à chaque itération, une nouvelle solution dans le voisinage de la solution courante. Si la nouvelle solution est meilleure, elle est acceptée, sinon elle est acceptée selon le critère de Metropolis, dans lequel la probabilité d'acceptation dépend de la dégradation de la fonction de coût (mesurée par la différence de coût entre la nouvelle solution et la solution courante) et d'un paramètre appelé *température*.

La méthode du recuit simulé a été conçue à l'origine pour la recherche de solutions dans des espaces discrets. Plusieurs travaux ont été menés pour l'adapter au cas de variables continues, citons quelques exemples :

- *Recuit simulé conventionnel* appelé aussi recuit de Boltzmann [German et German, 1984],

[4] Des travaux semblables, développés indépendamment à la même époque par V. Cerny [Cerny, 1985], ont été publiés en 1985.

- *Recuit simulé rapide* appelé aussi recuit de Cauchy [Szu et Hartley, 1987],
- *Recuit simulé très rapide* [Ingber et Rosen, 1992],
- *Recuit simulé généralisé* [Tsallis et Stariolo, 1996],
- Recuit simulé adaptatif [Ingber, 1995],
- *Recuit simulé combiné avec des méthodes de recherche directe* [Vetterling et al., 1994] [Cardoso et al., 1996, 1997] [Kvasnicka et Pospichal, 1997] [Hedar et Fukushima, 2002, 2004].
- *Recuit simulé amélioré* [Siarry et al., 1997],

1.2.1.1.1.3. Recherche tabou

L'idée de base de la recherche tabou a été initialement introduite dans [Glover, 1986] et indépendamment, une ébauche a été proposée dans [Hansen, 1986]. Une description de la méthode et de ses concepts est donnée dans [Glover et Laguna, 1997].

La recherche tabou partage avec l'algorithme du recuit simulé l'idée de guider la recherche locale pour éviter les optimums locaux. Contrairement au recuit simulé, elle utilise toutefois un critère d'acceptation déterministe. A chaque itération, la recherche tabou se déplace vers le meilleur voisin, même s'il détériore la fonction objectif. Ce critère d'acceptation pouvant faire cycler la recherche, en revenant vers des solutions visitées auparavant, les solutions acceptées sont stockées dans une liste tabou[5], les déplacements vers les solutions de cette liste étant interdits.

Plusieurs améliorations de la recherche tabou ont été proposées dans la littérature :

- *Recherche tabou robuste* [Taillard, 1991 et 1995] qui utilise une liste tabou de longueur variable et aléatoire, et une forme de mémoire à long terme.
- *Recherche tabou réactive* [Battiti et Tecchiolli, 1994 et 1995] dont l'idée principale est d'accroître la longueur de la liste tabou si le nombre de solutions revisitées est élevé, et de réduire la longueur de la liste si ce nombre est faible. Une autre particularité de la recherche tabou réactive est qu'elle effectue une série de déplacements aléatoires si elle se trouve bloquée dans une région de l'espace de recherche.

Bien que la recherche tabou ait été appliquée avec succès à une grande variété de problèmes d'optimisation combinatoire, les applications à l'optimisation continue restent très limitées comparativement à d'autres métaheuristiques comme le recuit simulé ou les algorithmes génétiques.

Parmi les adaptations aux espaces continus, on peut citer la recherche tabou continue améliorée [Chelouah et Siarry, 2000] dont l'idée de base est de rester le plus proche possible de l'algorithme de base de Glover. L'algorithme effectue une première phase de recherche globale (ou *diversification*) pour localiser les régions les plus prometteuses de l'espace. Puis, quand une telle région est trouvée, l'algorithme effectue

[5] En pratique, ce sont plutôt les mouvements qui ont permis d'atteindre ces solutions qui sont stockés et non les solutions elles mêmes.

une recherche locale[6] (ou *intensification*) dans cette région. Les déplacements dans l'espace de recherche sont effectués en utilisant un système de voisinage basé sur un ensemble de boules fermées concentriques, présentée dans [Siarry et Berthiau, 1997].

Certains auteurs proposent des adaptations à l'optimisation continue basées sur l'utilisation de méthodes de recherche directe :

- Recherche tabou combinée avec la méthode de Hookes et Jeeves [Al-Sultan et Al-Fawzan, 1997],
- Recherche tabou combinée avec la méthode de Nelder et Mead et avec la recherche par motifs [Hedar et Fukushima, 2003],
- Recherche tabou combinée avec l'algorithme du simplexe de Nelder-Mead [Chelouah et Siarry, 2004].

D'autres travaux sur l'application de la recherche tabou à l'optimisation continue sont [Cvijovic et Klinowski, 1995, 2002], [Franze et Speciale, 2001] et [Hu, 1992].

1.2.1.1.1.4. Autres méthodes à base de solution courante unique

D'autres métaheuristiques à base de solution unique existent :

- *la recherche par voisinage variable* [Mladenovic et Hansen, 1997] dont l'idée de base est de changer de structure de voisinage pour sortir d'un optimum local.
- *la recherche locale guidée* [Voudouris et Tsang, 1995] qui repose sur une modification dynamique de la fonction objectif, dans le but de pénaliser l'optimum local courant.
- GRASP (*Greedy Randomized Adaptive Search Procedure*) est une autre méthode basée sur la recherche locale répétée, dans laquelle chaque recherche locale est initialisée avec une nouvelle solution réalisable construite grâce à une procédure de construction « gourmande » et aléatoire [Feo et Resende, 1989, 1995]. Pour une revue de la littérature sur cette méthode, voir [Festa et Resende, 2002]. Plusieurs variantes et améliorations sont aussi présentées dans [Resende et Ribeiro, 2002].
- *la recherche locale réitérée* [Stützle, 1999a et 1999b] [Lourenço et al., 2002] [Martin et al., 1991]. Un inconvénient de la recherche locale répétée est qu'elle perd toute l'information sur la recherche passée, à chaque fois qu'une nouvelle recherche locale est démarrée. Plusieurs autres approches ont alors été proposées, permettant de tirer profit de l'historique de la recherche pour initialiser la recherche locale. Parmi ces méthode, on retrouve la *recherche locale réitérée*, dont l'idée de base consiste à utiliser les optimums locaux trouvés au cours des recherches locales précédentes, pour construire le point initial qui sera utilisé pour démarrer la suivante.

[6] Les deux phases de recherche locale et globale sont souvent désignées par les termes *intensification* et *diversification* dans la littérature.

1.2.1.1.2. Méthodes à base de population

Cette classe de métaheuristiques travaille explicitement avec une population de solutions. Nous pouvons distinguer dans cette classe de méthodes les algorithmes évolutionnaires, les algorithmes de colonies de fourmis, la recherche par dispersion, les algorithmes mémétiques, les algorithmes basés sur les essaims particulaires et ceux basés sur l'estimation de distributions.

1.2.1.1.2.1. Algorithmes évolutionnaires

Les algorithmes évolutionnaires sont des techniques d'optimisation itérative et stochastique, inspirées par des concepts issus de la théorie de l'évolution de Darwin. Un algorithme évolutionnaire simule un processus d'évolution sur une population d'individus, dans le but de les faire évoluer vers les optimums globaux du problème d'optimisation considéré. Au cours du cycle de simulation, trois opérateurs interviennent : recombinaison, mutation et sélection. La recombinaison et la mutation créent de nouvelles solutions candidates, tandis que la sélection élimine les candidats les moins prometteurs.

Quatre branches ont été identifiées comme faisant partie de la famille des algorithmes évolutionnaires : les algorithmes génétiques, les stratégies d'évolution, la programmation évolutionnaire et la programmation génétique (voir [Schoenauer et Michalewicz, 1997] pour une comparaison entre les algorithmes évolutionnaires).

Les algorithmes génétiques [Goldberg, 1989] [Holland, 1962 et 1975] s'inspirent des principes de la génétique. Pour cette raison, ces algorithmes accordent une grande importance à la distinction entre la représentation génétique d'un individu (*génotype*) et sa représentation réelle (*phénotype*). Une bonne introduction aux algorithmes génétiques est donnée dans [Michalewicz, 1999]. L'opérateur principal utilisé par les algorithmes génétiques pour la construction de nouvelles solutions est l'opérateur de *recombinaison* (appelé aussi *croisement*). Cet opérateur récupère des parties du génotype de deux ou plusieurs solutions (parents), qu'il combine pour construire un (ou plusieurs) nouveau génotype (enfant), héritant ainsi de certaines de leurs caractéristiques. L'utilisation de la recombinaison seule ne permet pas d'introduire du matériel génétique nouveau, puisque cet opérateur combine le matériel déjà présent dans la population. Pour remédier à ce problème, les algorithmes génétiques utilisent un deuxième opérateur, *la mutation*, comme opérateur secondaire permettant d'introduire de nouveaux gènes, inexistants dans la population.

Contrairement aux algorithmes génétiques, les stratégies d'évolution ([Rechenberg, 1965, 1973 et 1994] [Schwefel, 1965, 1977 et 1995]) reposent principalement sur l'opérateur de mutation, alors que la recombinaison, si elle est utilisée, l'est de manière secondaire. Le premier algorithme basé sur les stratégies d'évolution [Rechenberg, 1973] [Schwefel, 1981] travaille sur une population composée d'un seul individu, représenté par un vecteur de nombres réels. Cet individu subit une mutation gaussienne, addition d'une variable gaussienne de moyenne nulle et d'écart type σ. Le meilleur entre le père et l'enfant devient le père de la génération suivante. L'élément critique est le choix du paramètre σ : à l'origine, la règle de $1/5$ était utilisée pour ajuster le paramètre σ au cours de l'évolution (si plus de $1/5$ des mutations réussissent (resp. ne réussissent pas), accroître (resp. décroître) σ). Plus récemment, les

stratégies d'évolution reposent sur une population de plusieurs individus et sont désignées par $(\mu, \lambda) - \mathrm{ES}$ ou $(\mu + \lambda) - \mathrm{ES}$, où μ parents engendrent λ enfants. Les μ meilleurs individus qui deviendront les parents de la génération suivante sont choisis parmi les $\mu + \lambda$ (parents, plus enfants) dans la version élitiste $(\mu + \lambda) - \mathrm{ES}$, ou parmi les λ enfants, dans la version non élitiste $(\mu, \lambda) - \mathrm{ES}$.

La programmation évolutionnaire est une troisième branche des algorithmes évolutionnaires introduite initialement comme une technique d'intelligence artificielle, appliquée aux machines à états finis par [Fogel, 1962] [Fogel et al., 1966], et ensuite étendue par [Burgin, 1968, 1973] et [Atmar, 1976]. Bien qu'elle se soit développée dans des circonstances complètement différentes, elle est en fait assez semblable aux stratégies d'évolution. Par exemple, les individus sont représentés de la même manière que pour les stratégies d'évolution, et leurs opérateurs de mutation sont identiques. Cependant, la programmation évolutionnaire n'utilise aucun opérateur de recombinaison [Bäck et al., 1993] et repose sur un opérateur de sélection complètement différent.

Plus récemment, la programmation génétique [Koza, 1992] est apparue initialement comme sous-domaine des algorithmes génétiques, mais tend actuellement à être considérée comme une branche à part entière.

1.2.1.1.2.2. Algorithmes de Colonies de fourmis

L'origine de cette approche repose sur le comportement des fourmis lors de la recherche de nourriture. Ce comportement, décrit dans [Deneubourg et al., 1990], leur permet de trouver les plus courts chemins entre les sources de nourriture et leur nid. Pendant leurs déplacements entre le nid et les sources de nourritures, les fourmis déposent une substance sur le sol. Cette substance, appelée *phéromone*, sert à guider les fourmis vers les chemins les plus rapides pour arriver aux sources de nourriture. Les sentiers les plus courts sont les plus concentrés en phéromone. En effet, les phéromones s'évaporent au cours du temps et les fourmis qui empruntent les sentiers les plus courts arrivent rapidement à se procurer de la nourriture et rentrent au nid en déposant de la phéromone sur leur chemin de retour.

En s'inspirant du comportement ci-dessus, une famille d'algorithmes d'optimisation a été proposée par [Dorigo, 1992], puis [Dorigo et al., 1996 et 1999]. L'idée de base consiste à travailler sur une population de solutions, chacune correspondant à une fourmi. Une structure de donnée commune, partagée par toutes les fourmis, contient l'information sur la phéromone accumulée dans l'espace de recherche.

1.2.1.1.2.3. Recherche par dispersion (*scatter search*)

Proposée par [Glover, 1994], la recherche par dispersion combine les solutions appartenant à un *ensemble de référence* pour engendrer de nouvelles solutions. Des combinaisons linéaires d'un nombre variable de solutions sont effectuées pour créer de nouveaux candidats à l'intérieur et à l'extérieur de l'enveloppe convexe engendrée par l'ensemble de référence. Ce mécanisme de génération de nouvelles solutions est similaire à celui utilisé dans les algorithmes génétiques. Plusieurs solutions parentes peuvent être utilisées pour créer un enfant. Les enfants engendrés sont ensuite réparés ou améliorés et les meilleurs sont ajoutés à l'ensemble de référence. Une description

détaillée de cette métaheuristique est donnée dans [Glover et al., 2000] et [Laguna, 2002].

1.2.1.1.2.4. Algorithmes mémétiques

Les algorithmes mémétiques [Moscato, 1989 et 1993] s'inspirent de certains modèles d'adaptation dans la nature, qui combinent l'évolution adaptative de populations d'individus avec l'apprentissage des individus au cours de leur vie. Le terme « algorithme mémétique » trouve son origine dans le concept de « meme » introduit par Richard Dawkins [Dawkins, 1976], qui représente une unité d'information qui évolue avec les échanges d'idées entre individus. Une différence fondamentale entre les notions de gènes (manipulés par les algorithmes génétiques) et de « memes » est que ces derniers sont adaptés par la personne qui les transmet (en introduisant ses réflexions et déductions personnelles), alors que les gènes sont transmis tels quels.

Du point de vue optimisation, les algorithmes mémétiques sont des extensions des algorithmes évolutionnaires utilisant des procédures de recherche locale pour améliorer leurs individus. Ces algorithmes se retrouvent dans la littérature sous plusieurs autres noms : algorithmes génétiques hybrides [Fleurent et Ferland, 1994], recherche locale génétique [Merz, 2000], algorithmes évolutionnaires Baldwiniens [Ku et Mak, 1998], algorithmes évolutionnaires Lamarkiens [Morris et al., 1998].

1.2.1.1.2.5. Algorithmes basés sur un essaim de particules

L'optimisation par essaim de particules est une technique d'optimisation globale stochastique développée par [Kennedy et Eberhart, 1995], en s'inspirant du comportement social des individus qui ont tendance à imiter les comportements réussis qu'ils observent dans leur entourage, tout en y apportant leurs variations personnelles. Un ouvrage complet sur les racines sociales de cette métaheuristique et les techniques mathématiques mises en œuvre est [Kennedy et al., 2001].

L'optimisation par essaim de particules utilise une population de solutions potentielles (particules), qu'elle fait évoluer à l'aide des échanges d'informations (essaim) entre particules. En fait, chaque particule ajuste sa trajectoire vers sa meilleure position dans le passé et vers la meilleure position des particules de son voisinage [Kennedy, 1998]. La variante globale de ces algorithmes considère la totalité de l'essaim comme voisinage. Les particules profitent ainsi des découvertes et expériences antérieures de toutes les autres particules.

1.2.1.1.2.6. Algorithmes basés sur l'estimation de distribution

Récemment, plusieurs méthodes basées sur des modèles probabilistes ont été proposées [Larrañaga et al., 1999] [Pelikan et al., 2000]. Ces méthodes, connues sous le nom d'*algorithmes basés sur l'estimation de distribution*[7], construisent un modèle probabiliste pour les bonnes solutions, qu'elles utilisent pour guider la recherche. Contrairement aux algorithmes évolutionnaires qui construisent les nouvelles solutions en utilisant une information locale sur les bons individus à travers des opérateurs de

[7] Certains auteurs utilisent aussi les noms « Probabilistic Model Building Genetic Algorithms » ou « Iterated Density Estimation Algorithms » pour dénommer cette famille d'algorithmes

croisement et de mutation, les algorithmes basés sur l'estimation de distribution utilisent l'information globale contenue dans la population.

Plusieurs travaux ont été menés pour adapter cette famille d'algorithmes aux espaces continus :

- descente aléatoire avec apprentissage par des vecteurs gaussiens [Rudlof et Köppen, 1996],
- *apprentissage incrémental à base de population* [Sebag et Ducoulombie, 1998] [Servet et al., 1997],
- algorithme évolutionnaire basé sur l'estimation réitérée de densité [Bosman et Thierens, 2000],
- algorithme basé sur les distributions factorisées [Mühlenbein et Mahnig, 1999].

1.2.1.2. Méthodes statistiques

1.2.1.2.1. Plans d'expériences

Les plans d'expériences sont une première famille de méthodes statistiques très utilisées dans les expérimentations numériques. Cette famille de méthodes permet d'analyser un problème d'optimisation en boîte noire et d'aider au choix d'une solution, par énumération d'un nombre réduit de points de l'espace des solutions.

Parmi les techniques les plus utilisées pour la construction de plans d'expériences, on retrouve la méthode des plans factoriels [Montgomery, 2001]. On sélectionne, parmi les entrées du système, k facteurs pouvant avoir une influence sur les performances du système. Pour chacun d'entre eux, deux à trois valeurs critiques sont choisies. Si deux valeurs sont sélectionnées pour chaque facteur le plan est appelé un 2^k plan factoriel. De même, un 3^k plan factoriel correspond au choix de trois valeurs pour chacun des k facteurs. La construction d'un plan factoriel complet consiste à tester toutes les combinaisons de valeurs des facteurs. [Schaffer et al., 1989] présente une étude détaillée sur l'application d'un plan factoriel complet au paramétrage d'un algorithme génétique, 8400 expériences ayant été effectuées, avec 10000 réplications par expérience.

La méthode du plan complet peut rapidement devenir inutilisable, le nombre d'expériences croissant exponentiellement avec le nombre de paramètres. Une alternative consiste à utiliser un plan factoriel fractionnaire, qui se base sur une fraction d'expériences sélectionnées parmi celles du plan complet. Les plans d'expériences fractionnaires peuvent permettre d'étudier un grand nombre de facteurs, avec relativement peu d'expériences. L'une des approches utilisées pour la construction de plans fractionnaires est la méthode de Taguchi [Krottmaier, 1993].

En plus des plans factoriels, plusieurs autres techniques ont été décrites dans la littérature, telles que les plans d'hypercubes latins [Mckay et al., 1979], la suite échantillonnée de Hammersley [Kalagnanam et Diwekar, 1997] et les plans uniformes [Fang et Wang, 1994].

Des discussions sur l'utilisation des plans d'expérience et de l'analyse statistique pour les expérimentations numériques peuvent être trouvées dans [Barr et al., 1995] [Greenberg, 1990] [Crowder et al., 1978] [Hoaglin et Andrews, 1975].

Dans [Coy et al., 2001] les plans d'expériences et la méthode du gradient sont combinés pour déterminer un bon paramétrage pour les heuristiques de routage de deux véhicules. Les auteurs de [Laguna et Adenso-Diaz, 2002] proposent de combiner la méthode de Taguchi avec une méthode de recherche locale pour le réglage automatique des paramètres intervenant dans un algorithme d'optimisation. Les plans d'expérience permettent de focaliser la recherche locale sur les régions prometteuses de l'espace de recherche. D'autres références pour le paramétrage d'algorithmes sont [Van Breedam, 1995] pour une application au routage de véhicules et [Xu et al., 1998] pour l'application d'une méthode de recherche tabou à un problème de conception d'un réseau de télécommunication.

1.2.1.2.2. Tests statistiques

[Xu et al., 1998] proposent un algorithme basé sur les tests statistiques pour l'optimisation du réglage des paramètres intervenant dans une méthode de recherche tabou appliquée à un problème de conception de réseaux de télécommunication. Deux tests non paramétriques ont été utilisés :

- *le test de Friedman* [Hetimansperger, 1984], pour tester l'effet de la variation de la valeur d'un paramètre sur les résultats,
- *le test de Wilcoxon* [Devore, 1991], pour comparer deux réglages différents.

Ces tests sont incorporés au sein d'une procédure de recherche appelée « Tree growing and pruning method ». Cette procédure a été testée sur 19 instances du problème, les résultats ayant été amélioré dans 73% des cas (14 problèmes sur 19), par rapport au meilleur réglage connu pour les 5 paramètres analysés.

Parmi les travaux qui concernent la configuration de métaheuristiques, on retrouve aussi l'algorithme de course présenté dans [Birattari et al., 2002]. Cet algorithme a été inspiré par une classe de méthodes proposées en apprentissage automatique pour la résolution du problème de sélection de modèle [Maron et Moore, 1994] [Moore et Lee, 1994]. A partir d'un ensemble P_0 de solutions initiales, l'algorithme construit une séquence d'ensembles $P_0 \supseteq P_1 \supseteq P_2 \supseteq \ldots$, en éliminant à chaque étape certaines solutions considérées comme moins performantes. Le test de Friedmann est utilisé pour comparer les solutions.

1.2.1.3. Méthodes de recherche directe

Les méthodes de recherche directe forment une grande famille de méthodes d'optimisation qui fondent exclusivement leur recherche sur les valeurs de la fonction objectif.

1.2.1.3.1. Recherche par motifs

Les méthodes de recherche par motifs sont des méthodes itératives qui engendrent une suite de points $\{x_k\}$ à partir d'un point initial x_0 défini par l'utilisateur. Trois propriétés caractérisent cette famille d'algorithmes, relativement aux autres méthodes d'optimisation :

- à chaque itération, le nouveau point est sélectionné à partir d'un ensemble discret de points appelé *motif* [Dolan et al., 2000] ; plus précisément, un motif est représenté par une structure de treillis contenant le point courant,
- la valeur de la fonction objectif est la seule information utilisée,
- le motif est modifié au cours de l'optimisation, selon des règles permettant d'assurer la convergence à un point stationnaire.

Une description plus précise de cette famille d'algorithmes est proposée dans [Torczon, 1997].

Plusieurs variantes existent, la plus connue est la méthode de Hooke et Jeeves [Hooke et Jeeves, 1961] qui utilise une suite de mouvements de deux types : mouvements exploratoires et mouvements par motifs. Un mouvement exploratoire consiste à parcourir tous les axes de coordonnées de l'espace des solutions en ajoutant ou en soustrayant un pas à chaque variable x_i et à effectuer un déplacement vers le nouveau point s'il permet d'améliorer la fonction objectif. Un mouvement par motif est appliqué après un mouvement exploratoire en effectuant un déplacement le long de la direction définie par les deux solutions courantes, avant et après le mouvement exploratoire.

Pour prévenir la convergence prématurée de l'algorithme de Hooke et Jeeves, *Emery et al.* proposent la méthode de l'araignée qui effectue des déplacements le long de directions orthogonales dont l'orientation est choisie au hazard à chaque mouvement [Emery et al., 1966].

D'autres variantes existent, le lecteur intéressé peut consulter [Amitay, 2003].

1.2.1.3.2. Méthodes basées sur le simplexe

Un simplexe est un ensemble de n+1 points de $\mathbb{R}^n$. Un simplexe est dit régulier si les n+1 points sont mutuellement équidistants. Ces points forment un triangle équilatéral pour n=2, et un tétraèdre régulier pour n=3. En se basant sur les propriétés géométriques du simplexe, plusieurs méthodes ont été proposées dans la littérature :

- la méthode de Spendley et al. [Spendley et al., 1962],
- la méthode de Nelder et Mead [Nelder et Mead, 1965],
- la recherche multidirectionnelle [Torczon, 1989].

1.2.1.3.3. Méthodes avec des ensembles adaptatifs de directions de recherche

Ces méthodes tentent d'accélérer la recherche en se basant sur des directions construites en utilisant les informations sur les points visités. Parmi les méthodes appartenant à cette famille, on peut citer :

- la méthode de Rosenbrock [Bandler, 1969],
- la méthode DSC (Davies Swann et Campey) [Murray, 1972],
- la méthode de Powell [Powell, 1964].

1.2.2. Méthodes basées sur la méta-modélisation

Dans plusieurs problèmes rencontrés en pratique, l'évaluation des sorties du système à optimiser est trop coûteuse pour qu'une méthode basée sur l'évaluation exacte soit applicable. Une alternative consiste à utiliser des modèles approchés, appelés aussi méta-modèles [Kleijnen, 1987], pour représenter la relation entre les entrées et les sorties du système. Une fois construit, le méta-modèle est utilisé pour évaluer les sorties du système à optimiser, avec un coût de calcul considérablement réduit.

Les techniques de méta-modélisation ont été largement appliquées en pratique. Pour une revue de littérature des applications industrielles, voir [Barthelemy et Haftka, 1993], [Simpson et al., 1997] et [Sobieszczanski et Haftka, 1997].

Il existe deux approches pour l'application de techniques de méta-modélisation : l'approche directe, qui consiste à concevoir un modèle permettant de déterminer les sorties à partir des entrées, et l'approche inverse, dans laquelle le modèle construit permet de déterminer les entrées à partir des sorties (voir par exemple [Nasereddin et Mollaghasemi, 1999] pour une présentation de ces deux approches). L'avantage d'utiliser l'approche inverse est qu'il n'est plus nécessaire d'appliquer un algorithme d'optimisation au méta-modèle pour déterminer une solution : il suffit d'introduire les performances à atteindre en entrée du méta-modèle et de récupérer une solution en sortie.

Une grande variété de techniques de méta-modélisation existent, dont les principales sont décrites ci-dessous.

1.2.2.1. Réseaux de neurones artificiels

La technique des *réseaux de neurones artificiels* [Smith, 1993] s'inspire des capacités cognitifs et de traitement des données qui caractérisent les réseaux de neurones biologiques. Un réseau de base comprend trois couches : la *couche d'entrée*, le *couche cachée* et la *couche de sortie* (voir figure 1.2).

Chaque nœud de la couche d'entrée introduit dans le réseau la valeur d'une variable x_i sans aucun traitement sur cette donnée. Le nœud j de la couche cachée transforme les signaux d'entrée en un signal de sortie y_j en appliquant une *fonction d'activation* à la somme pondérée des entrées :

$$y_j = f\left(\sum_{i=1}^{n} w_{ij}\ x_i\right)$$

Les w_{ij} sont appelés *poids de connexion* et sont déterminés en estimant leurs valeurs à partir d'un ensemble de données, appelées *données d'apprentissage*. Un exemple de fonction d'activation est la fonction tangente hyperbolique :

$$f(u) = \frac{e^{u} - e^{-u}}{e^{u} + e^{-u}}.$$

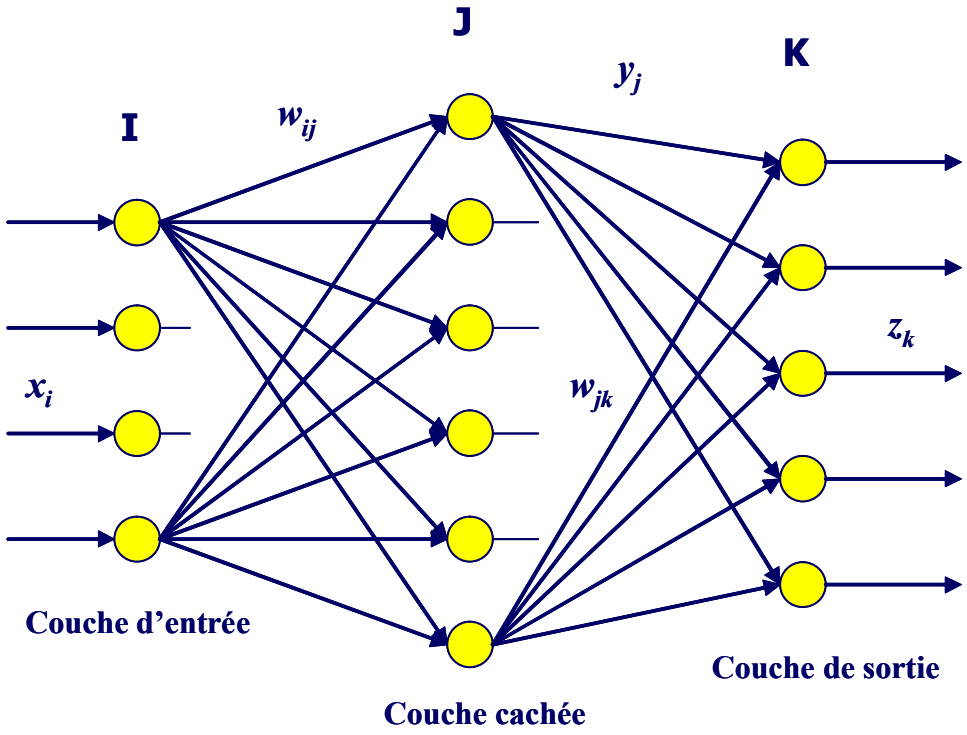


Figure 1.2 - *Architecture d'un réseau de neurones artificiels.*

Ces mêmes transformations sont aussi effectuées par les nœuds de sorties. Une fois les poids de connexion déterminés, le réseau est entraîné et peut être utilisé comme un modèle pour le système.

L'application des réseaux de neurones possède plusieurs avantages, comme la capacité d'approcher une fonction quelconque quelle que soit sa forme, ainsi qu'une grande capacité de généralisation et d'auto-adaptation aux changements dans l'environnement. Mais leur principal inconvénient réside dans le fait qu'ils nécessitent en général un volume important de données d'apprentissage pour atteindre une bonne précision.

1.2.2.2. Méthodologie des surfaces de réponse

La méthodologie des surfaces de réponse a été créée par [Box et Wilson, 1951] pour déterminer la combinaison d'entrées qui minimise la sortie d'un système réel. Elle se base principalement sur l'approximation polynomiale et la technique des moindres carrés. Plusieurs chercheurs l'ont appliquée à des problèmes complexes rencontrés dans le domaine de l'ingénierie [Unal et al., 1996] [Venter et al., 1996] [Simpson et al., 1997].

L'un des avantages de cette méthodologie est la possibilité d'identifier le degré d'importance des différents facteurs (ou entrées du système) directement à partir des coefficients du modèle polynomial construit. Il est alors possible de sélectionner les variables critiques et de réduire la dimensionnalité du problème, ce qui est très utile pour les problèmes avec un nombre élevé de variables. Cependant, cette méthodologie possède un sérieux inconvénient, lorsque la fonction à optimiser est fortement non

linéaire. Dans ce cas, il est en effet nécessaire d'utiliser des polynômes de degré élevé, ce qui est très coûteux, car le nombre de coefficients à estimer s'accroît exponentiellement avec le degré du polynôme.

1.2.2.3. Méthode de Kriging

La méthode de Kriging est une technique d'interpolation permettant de prédire les valeurs inconnues d'un processus aléatoire (voir l'ouvrage classique traitant de la méthode de Kriging [Cressie, 1993]). Elle a été développée à l'origine par l'Ingénieur des Mines sud africain D. G. Krige, dans les années 50, puis améliorée avec l'aide du mathématicien français G. Matheron de l'École des Mines. Dans la même période, en météorologie L. Gandin travaillait sur des idées similaires, sous la désignation d'« *optimum interpolation* ». La méthode de Kriging est aussi connue sous les noms de *modèles à corrélation spatiale* [Barton, 1998] et DACE (*Design and Analysis of Computer Experiments*) [Sacks et al., 1989]. Récemment, cette méthode est devenue très populaire dans la méta-modélisation de simulations déterministes [Simpson et al., 1998] [Trosset et Torczon, 1997] [Karimi et al., 1996].

1.2.2.4. Autres techniques de méta-modélisation

D'autres techniques de méta-modélisation sont :

- *la régression polynomiale*, qui a été appliquée avec succès par plusieurs auteurs [Engelund et al., 1993] [Unal et al., 1996]. Elle est toutefois inadaptée à l'approximation de surfaces complexes [Barton, 1993].
- *La régression adaptative multivariée de splines* [Friedman, 1991], qui est une technique non paramétrique, c'est-à-dire qu'elle ne suppose a priori aucune forme analytique pour la fonction à approcher. Comme la régression polynomiale, elle est inadaptée à l'approximation de surfaces complexes [Sage, 2003].
- *Les fonctions à base radiale* [Hardy, 1971] [Dyn et al., 1986], qui ont été utilisées avec succès pour l'approximation de fonctions déterministes et stochastiques [Powell, 1987].

1.3. Application au paramétrage d'un « générateur d'ordonnancement »

Il existe une vaste littérature sur les techniques utilisées pour la résolution de problèmes d'ordonnancement, tirant partie des avancées technologiques dans des domaines comme la programmation mathématique, l'intelligence artificielle, la logique floue ou l'apprentissage automatique. Pour une revue de littérature des techniques d'optimisation appliquées à l'ordonnancement voir [Choi, 2000] [Jain et Meeran, 1998] [Jones et Rabelo, 1998].

On peut séparer les approches utilisées pour la résolution de problèmes d'ordonnancement en deux catégories : les approches par représentation explicite des solutions et ceux par représentation implicite.

La première catégorie d'approches (avec représentation explicite des solutions) consiste à construire un ordonnancement en se basant sur un modèle spécifique du type de problème considéré. Parmi les méthodes appartenant à cette catégorie on retrouve les algorithmes génétiques basés sur une *représentation explicite* des solutions [Nakano et Yamada, 1991] [Bruns, 1993].

Contrairement à la première, la seconde catégorie d'approches (avec représentation implicite des solutions) consiste à représenter un ordonnancement par un ensemble de paramètres qui sont les entrées d'une procédure indépendante permettant de construire un ordonnancement, par exemple un simulateur ou un logiciel d'ordonnancement du marché. Nous appellerons une telle procédure un *« générateur d'ordonnancement »*. La procédure d'ordonnancement est donc vue comme une boîte noire, avec des entrées (paramètres du générateur) et des sorties (indicateurs permettant d'évaluer la qualité de l'ordonnancement construit par le générateur). Nous nous intéresserons par la suite exclusivement aux méthodes basées sur une représentation implicite des solutions, appelées aussi *approches de type boîte noire*.

Dans ce qui suit nous présenterons quelques applications structurées selon la classification proposée précédemment.

1.3.1. Approches basées sur l'évaluation exacte

Plusieurs travaux dans la littérature des algorithmes génétiques peuvent être considérés comme des approches de type boîte noire appliquées à l'ordonnancement. La plupart de ces travaux sont basés sur un codage particulier des solutions, appelé *représentation à base de règles de priorité.* L'ordonnancement est représenté de manière implicite sous forme de chromosome dont les gènes correspondent à des heuristiques utilisées pour construire un ordonnancement. Cette approche, appelée aussi *méthode par combinaison d'heuristiques* (HCM), a été publiée simultanément et indépendamment dans [Norenkov, 1994] et [Fang et al., 1994]. Dans [Norenkov et Goodman, 1997], un algorithme évolutionnaire basé sur une représentation HCM est appliqué à un problème d'ordonnancement de type « flow shop[8] ». Parmi les travaux basés sur cette approche, on peut citer [Dorndorf et Pesch, 1995] et [Hart et Ross, 1998].

Les auteurs de [Sauer et al., 1997] et [Weigert et al., 1998] proposent un système d'optimisation basé sur un simulateur pour les processus manufacturiers discrets. Ce simulateur (appelé ROSI) offre la possibilité d'évaluer un système à évènements discrets à partir de fonctions de coût définies par l'utilisateur, et permet aussi de spécifier les paramètres à faire varier à l'entrée du système. Deux types de paramètres peuvent être définis : *permutation* pour la modélisation des séquences, et *réglage* pour la modélisation des paramètres qui possèdent une valeur numérique réglable. L'optimisation est effectuée au moyen d'une approche d'optimisation par simulation, dans laquelle deux modules indépendants communiquent : un module d'optimisation, qui recherche de nouvelles valeurs pour les paramètres et les injecte à l'entrée du système, et un module de simulation, qui évalue les fonctions de coût, qui sont ensuite récupérées par le module d'optimisation. Des algorithmes d'optimisation du type recuit

[8] Un problème d'ordonnancement de type « flow shop » suppose que les tâches doivent passer sur le même ensemble de machines, avec un ordre de passage identique sur les machines.

simulé et algorithmes génétiques ont été implémentés. Une procédure d'optimisation a aussi été proposée dans le cas d'un ordonnancement multi-objectif [Weigert et al., 2000]. Cette procédure repose sur une agrégation par somme pondérée des objectifs normalisés, sur une normalisation dynamique (au cours de l'optimisation) des objectifs, ainsi que sur une méthodologie pour la réduction du nombre d'objectifs.

L'étude de [Muhl et al., 2003] concerne l'ordonnancement de la production dans une usine de production automobile. Les véhicules passent successivement à travers trois ateliers : ferrage, peinture et montage. Une première séquence du flux de véhicules est définie à l'entrée du premier atelier (ferrage). A l'intérieur de chaque atelier, un algorithme d'ordonnancement local (ARI : Algorithme de Re-séquencement Intermédiaire) permet ensuite de réadapter la séquence de véhicules aux contraintes et perturbations locales. Au niveau de chaque atelier, l'ARI offre la possibilité à l'utilisateur de spécifier un poids pour chaque contrainte à satisfaire. Ce poids quantifie l'importance accordée à la satisfaction de la contrainte correspondante relativement aux autres. Afin de satisfaire les objectifs globaux (principalement les délais client) les auteurs proposent de coordonner les ARI au niveau des trois ateliers en ajustant leurs 9 paramètres (poids associés aux 9 contraintes locales). Plusieurs métaheuristiques ont été utilisées pour optimiser les valeurs de ces paramètres : le recuit simulé, la descente aléatoire multistart, l'algorithme génétique et l'échantillonnage aléatoire. L'évaluation de la fonction objectif est obtenue par simulation, grâce au logiciel ARENA. Les résultats obtenus sur l'exemple testé montrent des améliorations des performances entre 12 % et 41 % par rapport à la situation initiale.

Les auteurs de [Paris et Pierreval, 2001] proposent un algorithme évolutionnaire distribué pour configurer les systèmes de production de type kanban. Plusieurs processeurs sont utilisés simultanément pour ajuster les valeurs des différents paramètres qui ont une influence sur les performances du système : nombre de kanban entre machines, taille des lots, taille des stocks de sécurité, règles d'ordonnancement,… La parallélisation de l'approche a permis d'accélérer la recherche de l'optimum et d'élargir l'espace de recherche exploré.

Parmi les travaux consacrés à la configuration de systèmes de type kanban, on peut citer aussi [Becker et Szczerbicka, 1998], qui utilisent un couplage entre un outil d'optimisation global appelé REMO (REsearch Model Optimization) et un modèle de simulation basé sur les réseaux de Petri. L'outil REMO est basé sur une approche hybride, qui combine des stratégies d'optimisation globales (algorithmes génétiques ou recuit simulé) avec des stratégies d'optimisation locale basées sur la méthode de recherche directe de Hookes et Jeeves [Syrjakow et Szczerbicka, 1994].

1.3.2. Approches basées sur la méta-modélisation

Dans les approches classiques utilisées en ordonnancement, une seule règle de priorité est appliquée à toutes les machines dans un système manufacturier. Baek et al. proposent d'identifier une règle de priorité adéquate pour chaque machine [Baek et al., 1998] : c'est ce qu'ils appellent l'adaptabilité spatiale, en opposition avec l'adaptabilité temporelle, qui consiste à changer de règle de priorité au cours du temps. Les règles de priorité prennent alors la forme de sommes pondérées de critères, les poids étant déterminés par apprentissage à partir d'une exécution répétée de simulations. Une approche par les plans d'expériences de Taguchi est utilisée pour déterminer des poids

efficaces et robustes. La robustesse concerne la variation des performances due à des perturbations dans le processus manufacturier et son environnement [Byrne et Taguchi, 1987]. Les résultats obtenus par la méthode proposée sur un atelier de fabrication de composants pour semi-conducteurs ont été jugés nettement meilleurs que ceux obtenus avec les méthodes classiques utilisées pour la sélection de règles de priorité.

Un inconvénient de la méthodologie des surfaces de réponses est qu'elle n'est pas applicable quand les entrées du système sont qualitatives. Plusieurs travaux [Shang and Tadikamalla, 1993] [Shang, 1995] [Shang and Tadikamalla, 1998] [Chiadamrong, 2003] proposent alors de combiner la méthodologie des surfaces de réponses avec la méthode des plans d'expériences de Taguchi pour optimiser les entrées de systèmes manufacturiers, en tirant profit des avantages et inconvénients de chacune des deux méthodes.

Dans [Grabot et Geneste, 1994], le paramétrage du logiciel d'ordonnancement industriel « SIPAPLUS » est considéré en utilisant trois méthodes différentes : un réseau de neurones, un système expert et la méthode des plans d'expériences de Taguchi. L'approche proposée consiste à régler les paramètres utilisés par une nouvelle règle d'ordonnancement intégrée dans une version expérimentale de SIPAPLUS. Cette nouvelle règle est construite en combinant plusieurs règles simples avec une technique basée sur les sous-ensembles flous [Zadeh, 1965]. Une étude comparative entre les trois méthodes expérimentées a permis de constater la supériorité de l'approche par le réseau neuronal, relativement aux deux autres. Comparés aux résultats procurés par les règles standard utilisées par SIPAPLUS, les résultats du réseau neuronal étaient meilleurs dans plus de 60 % des cas étudiés.

Les auteurs de [Lereno et al., 2001] proposent une approche pour simplifier l'utilisation des logiciels d'ordonnancement en introduisant un apprentissage systématique de leur utilisation, en capitalisant les connaissances mises en œuvre lors des premières utilisations, afin de choisir les meilleurs calendriers d'opérateurs par l'analyse des tâches à ordonnancer et de configurer automatiquement le logiciel (choix des bons paramètres). L'apprentissage a été effectué en utilisant deux techniques différentes : les arbres de décision, et les réseaux de neurones à fonction radiale de base (RBF). Ces méthodes ont été appliquées au paramétrage du logiciel d'ordonnancement « PREACTOR » sur un exemple de problème d'ordonnancement. Les résultats obtenus par le réseau RBF, pour cet exemple, ont été meilleurs que ceux obtenus par les arbres de décision.

1.3.3. Autres approches

L'auteur de [Rogers, 2002] propose l'utilisation d'*OptQuest* pour l'optimisation de certaines variables intervenant dans la modélisation de systèmes de production avec l'outil de simulation *ARENA*. *OptQuest* est basé sur une approche hybride combinant la recherche tabou, la recherche par dispersion (*scatter search*) et un réseau neuronal. Ce dernier est utilisé comme model prédictif pour accélérer la recherche, en évitant d'évaluer par simulation certaines solutions dont la valeur prévisible est mauvaise. L'utilisateur choisit les paramètres de contrôle (sélectionnés parmi les variables du modèle *ARENA*, ainsi que les capacités de ses ressources) qu'il veut ajuster, et fournit pour chacun des bornes supérieures et inférieures. Ensuite, l'utilisateur spécifie la fonction objectif à optimiser, qui est construite à partir d'un ensemble de statistiques

définies dans le modèle. *OptQuest* offre aussi la possibilité d'exploiter des fonctionnalités additionnelles, comme restreindre l'espace de recherche, ou imposer des conditions sur les solutions. Plus précisément, l'utilisateur peut définir des contraintes linéaires entre les paramètres de contrôle choisis, et imposer des bornes supérieures ou inférieures à certaines mesures de performance additionnelles.

L'étude de [Huyet et Paris, 2003] traite du problème général de l'optimisation et de l'analyse des paramètres intervenant dans les systèmes de production. Les objectifs sont :

- de déterminer une combinaison de paramètres permettant d'optimiser les performances d'un système de production,
- d'expliquer l'interaction entre les paramètres et leur influence sur les performances de ce système.

Une méthodologie dénommée *optimisation intelligente* est proposée, basée sur l'utilisation conjointe d'un algorithme évolutionnaire pour l'aspect « optimisation de la combinaison de paramètres » et d'une méthode d'apprentissage, pour l'aspect « analyse des paramètres ». L'algorithme évolutionnaire est connecté à deux modules :

- un module de simulation, qui permet d'évaluer les solutions (combinaisons de paramètres, à l'entrée du système de production),
- un module d'apprentissage, qui permet de tirer profit des solutions visitées, pour diriger l'optimisation vers les régions prometteuses et de construire une connaissance sur l'espace des paramètres, en se basant sur les performances des solutions déjà visitées.

Une étape d'apprentissage est effectuée toutes les k générations de l'algorithme évolutionnaire. Les solutions engendrées au cours de l'optimisation sont utilisées comme base d'exemples pour l'algorithme d'apprentissage qui, en se basant sur la connaissance acquise, introduit de nouvelles solutions prometteuses dans la population courante, de façon à focaliser la recherche sur les régions intéressantes et à accélérer ainsi la convergence de l'algorithme évolutionnaire. La méthode d'apprentissage choisie est basée sur les graphes d'induction [Zighed et Rakotomalala, 2000] et l'algorithme C4.5 développé par [Quinlan, 1993].

Un système de production du type kanban, constitué de 5 machines et 3 types de produits, a été utilisé pour illustrer l'approche. Trois types de paramètres ont été considérés : les règles de priorité, les tailles des lots et le nombre de kanban. La performance du système est évaluée avec une somme pondérée des retards des ordres de fabrication et de l'encours (*Work In Progress*).

1.4. Conclusion

Bien que les techniques présentées dans ce chapitre soient destinées au modèle de l'optimisation en boîte noire déterministe et monocritère, la plupart possèdent des adaptations au cas où :

- *les sorties sont aléatoires :* les méthodes utilisées sont souvent référencées sous le terme *optimisation par simulation* (voir par exemple [April et al., 2003]),

- *les sorties sont multiples :* la plupart des problèmes d'optimisation rencontrés en pratique sont de nature multicritère, en particulier les problèmes d'ordonnancement [T'Kindt et Billaut, 2002]. Plusieurs approches ont été proposées dans la littérature des métaheuristiques (voir par exemple [Talbi, 2000] pour un état de l'art),
- on mène une optimisation « parallèle » en distribuant l'optimisation sur plusieurs processeurs, de manière à réduire le temps d'optimisation [Cung et al., 2002].

Malgré l'immensité des travaux de recherche dédiés aux algorithmes d'optimisation en boîte noire, celle-ci reste à l'heure actuelle un domaine de recherche très actif, à cause de son grand intérêt pratique et de la multitude de difficultés liées à l'application de ces algorithmes à des problèmes de taille réelle. Dans le cas de notre problématique de paramétrage d'un logiciel d'ordonnancement les principales difficultés sont :

- le nombre important de paramètres (plusieurs dizaines) à régler,
- la nature continue des paramètres,
- l'évaluation de la fonction objectif, nécessitant la réalisation d'un ordonnancement, ce qui est très coûteux en temps de calcul,
- la problématique de choix d'un nombre réduit de paramètres (voir le chapitre précédent) pour éviter l'explosion combinatoire, réduire l'ordonnancement et améliorer l'interprétation des résultats.

A cause de toutes ces difficultés, les méthodes décrites dans ce chapitre ne sont pas adaptées si elles sont utilisées seules, et de plus en plus de travaux tentent de proposer des approches hybrides en combinant plusieurs techniques différentes.

L'idée que nous proposons ici consiste à combiner deux approches différentes :

- les métaheuristiques,
- des techniques de sélection de paramètres, inspirées de certaines méthodes utilisées en statistiques (sélection de variables en régression linéaire) et en apprentissage automatique (sélection d'attributs).

L'approche proposée (qui sera exposée en détail dans le chapitre suivant) peut ainsi résoudre les deux problématiques de sélection de paramètres et de réglage de manière simultanée.

Chapitre 2

Élaboration de métaheuristiques pour la sélection et l'optimisation de paramètres d'un logiciel d'ordonnancement

2.1. Introduction

L'application d'un logiciel d'ordonnancement à une instance donnée peut être vue comme une boîte noire (voir figure 2.1), avec :

- *en entrée :* un ensemble de *paramètres* $x_1, x_2, ..., x_N$ du logiciel (pour simplifier, la valeur d'un paramètre x_i sera notée x_i également),
- *en sortie :* un ensemble de critères permettant de juger la qualité du résultat (ordonnancement) obtenu. Ces critères sont généralement représentés par des fonctions numériques $f_1, f_2, ..., f_M$, appelées *fonctions objectifs* ou *indicateurs de performance*.

De nombreuses études montrent qu'il est nécessaire de bien configurer les entrées (ou paramètres) du logiciel si l'on veut atteindre de bonnes performances en sortie du logiciel utilisé [Grabot et al., 1994] [Lereno et al., 2001]. Pour diverses raisons liées à la complexité des logiciels d'ordonnancement et des cas industriels rencontrés en pratique, cette tâche de « configuration des entrées » est difficile à effectuer manuellement en utilisant l'expérience acquise et les connaissances sur le problème industriel à résoudre (voir la description de cette problématique dans le chapitre introduction générale). Divers travaux ont été menés pour automatiser cette tâche, par rapport à un cas industriel donné, en général en présence d'un nombre relativement limité de paramètres (voir le chapitre 1 pour un état de l'art).

Dans ce chapitre nous présenterons l'approche nouvelle que nous proposons pour le paramétrage automatique d'un logiciel d'ordonnancement industriel, basée sur l'utilisation conjointe de deux familles de méthodes : les métaheuristiques et les stratégies de sélection de paramètres.

Figure 2.1 - *Logiciel d'ordonnancement vu comme une boîte noire.*

Deux problèmes seront abordés :

- *un problème de sélection (ou choix) de paramètres*, consistant à choisir un sous ensemble (le plus petit possible) de paramètres,
- *un problème d'optimisation (ou de réglage)*, consistant à ajuster les valeurs des paramètres choisis, de façon à optimiser la valeur des indicateurs de performance $f_1, f_2, ..., f_M$.

Ces deux problèmes sont très liés, et ne peuvent être résolus indépendamment l'un de l'autre. Les choix effectués au moment de la sélection de paramètres conditionnent en effet fortement la qualité des réglages obtenus au niveau du problème d'optimisation. Pour cette raison, nous proposerons dans ce chapitre une approche permettant de les résoudre de façon simultanée.

Dans la deuxième partie, nous donnerons des définitions et émettrons quelques hypothèses de départ, puis nous présenterons notre approche d'optimisation multicritère. Celle-ci comprend une étape de calcul réalisée à l'aide de métaheuristiques. Un grand nombre de métaheuristiques existent dans la littérature (voir par exemple [Blum et Roli, 2001] [Voß, 2001]) ; celles que nous avons choisies sont adaptées à deux caractéristiques du logiciel d'ordonnancement *Ortems* :

- le nombre de paramètres est important,
- l'espace des valeurs des paramètres est continu.

Les métaheuristiques choisies sont décrites dans la partie 4, avec une présentation de la démarche employée pour prendre en compte la sélection de paramètres au cours de l'optimisation. Cette démarche repose sur plusieurs stratégies de sélection de paramètres qui sont décrites dans la partie 3.

2.2. Notations, définitions et hypothèses

Une *solution* est un vecteur $x=(x_1,x_2,...,x_N)$, où x_i valeur du i^{ème} paramètre prend ses valeurs dans un domaine X_i de $\mathbb{R}$ (ensemble des nombres réels). On notera X l'espace des solutions $X_1 \times X_2 \times \cdots \times X_N$.

Nous supposerons satisfaites les hypothèses suivantes :

(H_1) Chaque paramètre x_i prend ses valeurs dans un intervalle $X_i = [\underline{x}_i, \overline{x}_i]$.

(H_2) Chaque indicateur f_j est une fonction numérique définie sur l'espace des solutions X,

(H_3) Les fonctions f_j sont de type « boîtes noires déterministes » (voir chapitre 1 pour une définition de ce type de fonctions).

Notons A l'algorithme choisi pour ajuster les valeurs des paramètres en entrée du logiciel. On peut définir plusieurs concepts liés au choix de paramètres au sein de A : *désactivation*, *élimination* et *sélection* de paramètres. Le premier (*désactivation*) repose sur l'idée de figer les valeurs de certains paramètres pour la durée totale d'exécution de l'algorithme. Les deux autres concepts (l'*élimination* et la *sélection*) caractérisent les paramètres qui, selon leurs valeurs, peuvent contribuer ou non à la construction de l'ordonnancement par le logiciel. Dans ce cas, un paramètre est dit *éliminé* si sa valeur implique qu'il n'est pas utilisé par le moteur d'ordonnancement ; il est dit *sélectionné* dans le cas contraire. Si l'on considère la problématique de paramétrage dans le logiciel *Ortems*, les paramètres à régler sont des poids affectés à des règles d'ordonnancement dans une somme pondérée. Si un poids est nul, la règle correspondante n'est plus utilisée par le moteur d'ordonnancement du logiciel. Dans ce cas particulier, nous dirons qu'un paramètre est éliminé, si sa valeur est nulle.

Nous ferons donc les deux hypothèses suivantes :

(H_4) tous les paramètres peuvent être éliminés en leur affectant une valeur nulle,

(H_5) la sélection d'un paramètre qui était éliminé a un coût.

L'hypothèse (H_5) est importante pour justifier l'objectif de sélection d'un nombre minimum de paramètres dans notre problématique. Par exemple, dans le cas du logiciel d'ordonnancement *Ortems*, la sélection d'un paramètre implique plusieurs types de coûts :

- cette sélection peut s'interpréter comme l'utilisation d'une règle supplémentaire dans l'ordonnancement, ce qui induit un coût lié à l'augmentation de la durée d'ordonnancement[9].
- Un deuxième coût est lié à la prise en compte d'un plus grand nombre de paramètres dans l'algorithme d'optimisation[10]. Ce coût est d'autant plus

[9] Durée nécessaire pour réaliser un ordonnancement.

important que le nombre de paramètres est élevé ou que l'évaluation d'une solution est plus coûteuse.

- Le dernier coût est lié à la difficulté accrue d'interprétation de la solution recherchée.

Pour simplifier les notations, nous supposerons également que :

$(\mathbf{H_6})$ **les fonctions f_j sont à minimiser,**

et (H_1) sera remplacée (moyennant une transformation linéaire[11]) par :

$(\mathbf{H_1'})$ **chaque paramètre x_i est un nombre réel à valeurs dans $[0,1]$.**

2.3. Cadre général de l'approche

Le problème de réglage et de sélection des paramètres d'un logiciel d'ordonnancement revient à déterminer un vecteur $x=(x_1,x_2,...,x_{\mathrm{N}})$ (où x_i est la valeur du i[ème] paramètre) qui minimise sur l'espace des paramètres $[0,1]^{\mathrm{N}}$ les fonctions f_i, $i=1,2,...,\mathrm{M}$, (représentant les sorties du logiciel) et la fonction S définie par :

$S(x)=$ nombre de paramètres sélectionnés (i.e. non nuls[12]) dans x.

En d'autres termes, notre problème consiste à résoudre le programme multi-objectif :

$$\underset{x\in[0,1]^N}{Min}\ f_1(x),f_2(x),...,f_M(x),S(x)$$

où f_i, $i=1,2,...,\mathrm{M}$, sont les sorties du logiciel d'ordonnancement (voir la partie précédente).

Plusieurs approches existent pour la résolution d'un tel programme, en fonction du type de mise en œuvre des préférences de l'utilisateur [Andersson, 2000] :

- *Aucune articulation des préférences :* ce type de méthodes n'utilise aucune information sur les préférences de l'utilisateur[13]. Des exemples sont la formulation Min-Max et la méthode du critère global (voir [Steuer, 1986]).
- *Articulation a priori des préférences :* il s'agit de ramener le problème multi-objectif à un problème mono-objectif en utilisant une agrégation des fonctions f_j. Selon la méthode d'agrégation utilisée, on retrouve plusieurs approches : approches par les sommes pondérées, approches non linéaires, approches par la

---

[10] Algorithme permettant d'ajuster les valeurs des paramètres.

[11] La transformation est donnée par $h(x_i)=(x_i-\underline{x}_i)/(\overline{x}_i-\underline{x}_i)$.

[12] Cf. hypothèse (H_4).

[13] Tout au long de ce chapitre, nous ferons référence au terme « utilisateur » pour désigner un ou plusieurs décideurs, dont les objectifs doivent être pris en compte pour paramétrer le logiciel d'ordonnancement.

logique floue, théorie de l'utilité, … (voir [Andersson, 2000] pour une revue générale de ces méthodes).

- *Articulation progressive des préférences (ou méthode interactive) :* l'utilisateur interagit avec le programme d'optimisation durant le processus d'optimisation. Parmi les méthodes utilisant ce type d'approche, on retrouve les méthodes de STEM, Steuer, Vanderpooten, … (voir par exemple [Roy et Bouyssou, 1993]).
- *Articulation a posteriori des préférences :* l'algorithme d'optimisation fournit un ensemble de solutions efficaces, à partir desquelles l'utilisateur choisit la solution qu'il préfère. Des exemples sont les approches par exécution répétée, le recuit simulé multiobjectif, les algorithmes génétiques multiobjectifs, … (voir [Talbi, 2000]).

L'approche que nous avons adoptée ici est une approche à articulation progressive des préférences. Les raisons en sont les suivantes :

- en pratique, les objectifs d'ordonnancement sont souvent complexes et mal définis initialement,
- les objectifs d'ordonnancement peuvent changer au cours du temps [Rodammer et White, 1988] [Smith, 1992],
- les autres approches présentent plusieurs inconvénients :

  - les approches sans articulation de préférences ne sont pas adaptées, car elles ne prennent pas en compte l'importance de certains critères relativement aux autres,

  - les approches avec articulation a priori exigent une connaissance parfaite des préférences de l'utilisateur, ce qui est rarement le cas dans les problèmes d'ordonnancement rencontrés en pratique,

  - les approches à articulation a posteriori des préférences sont plus difficiles à mettre en œuvre, et plus coûteuses en temps de calcul.

2.3.1. Approche multicritère

La figure 2.2 montre les différentes étapes de notre approche multicritère. La solution est obtenue à travers un processus itératif ; à chaque itération, l'utilisateur analyse la meilleure solution obtenue jusqu'ici. S'il est satisfait, le processus prend fin. Dans le cas contraire, il effectue des changements dans ses préférences, puis lance une optimisation approchée[14].

2.3.1.1. Étape de définition des préférences

A l'initialisation et à la fin de chaque optimisation, l'utilisateur change ses préférences, au moyen des actions suivantes :

[14] Un algorithme permettant de trouver une *bonne* solution, compte tenu du temps disponible

- définition du sous-ensemble[15] I d'indicateurs à optimiser et spécification des valeurs des poids w_i associés à ces fonctions (ces poids matérialisent l'importance accordée à un objectif relativement aux autres).
- Définition du sous-ensemble J d'indicateurs pour lesquels l'utilisateur souhaite spécifier des contraintes du type : $f_j \leq b_j, j \in \mathrm{J}$, et spécification des valeurs des $b_j, j \in \mathrm{J}$ (les b_j peuvent s'interpréter comme un niveau de performance minimum exigé par l'utilisateur sur l'indicateur j).
- Désactivation de certains paramètres (définition du sous-ensemble L des paramètres à désactiver).

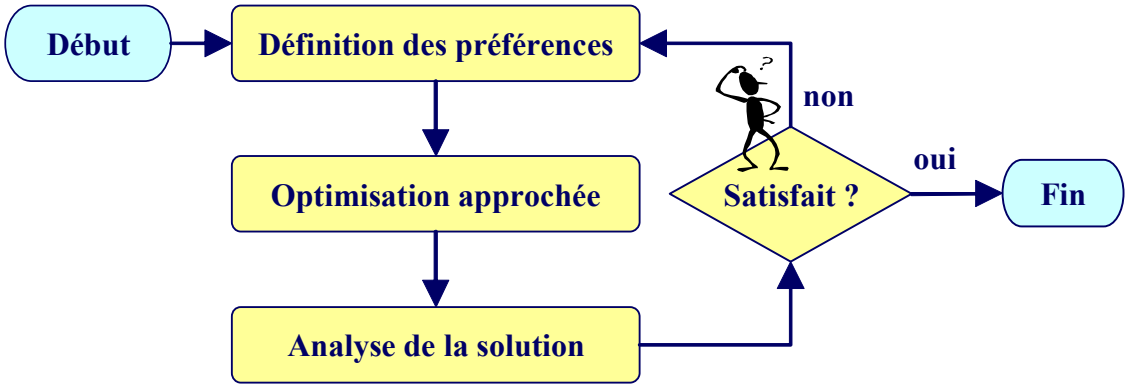


Figure 2.2 - *Principe général de l'approche.*

Pour simplifier, les ensembles I et J sont considérés disjoints (cette hypothèse n'est pas nécessaire dans l'approche générale présentée ici). En d'autres termes, pour chacun des indicateurs choisis, l'utilisateur peut soit imposer une borne supérieure à ne pas dépasser, soit attribuer un poids relativement aux autres indicateurs, mais pas les deux en même temps. La modification des valeurs des poids associés aux fonctions f_i permet à l'utilisateur de changer l'importance relative qu'il attribue aux différents objectifs. De plus, si une solution obtenue en lançant une optimisation est mauvaise du point de vue de la satisfaction de certains objectifs, et bonne du point de vue de certains autres, en imposant des contraintes strictes sur ces derniers, l'utilisateur peut lancer une nouvelle optimisation, sans risquer de trop les dégrader.

2.3.1.2. Étape d'optimisation approchée

A la k^{ème} itération, l'étape d'optimisation approchée consiste à résoudre de façon approchée le programme mathématique bi-objectif $\left(\mathrm{P}_1^{(k)}\right)$ suivant :

[15] Ce sous-ensemble est défini à partir de l'ensemble des indicateurs de performance de sortie du logiciel à paramétrer.

$$\left(\mathrm{P}_1^{(k)}\right)\begin{cases} Min\, f(x), S(x) \\ x \in \mathrm{X_k} \\ f_j(x) \le b_j,\ j \in \mathrm{J} \end{cases}$$

où X_k est un sous-ensemble de l'espace des paramètres $[0,1]^N$, obtenu en désactivant certains paramètres (dans l'étape de définition des préférences) :

$$\mathrm{X}_k = \left\{(x_1, x_2, ..., x_N) \in [0,1]^N ;\ x_i = x_i^0, i \in \mathrm{L}\right\},$$

L est l'ensemble des indices des paramètres désactivés et x_i^0 est la valeur du paramètre x_i au point initial.

Les fonctions objectifs de $\left(\mathrm{P}_1^{(k)}\right)$ sont :

- S : nombre de paramètres sélectionnés dans une solution,
- $f = \sum_{i \in I} w_i \bar{f}_i$: somme pondérée des sorties (normalisées) du logiciel à paramétrer, w_i est un poids associé au critère i ($0 < w_i \le 1$ et $\sum_{i \in I} w_i = 1$), qui matérialise l'importance relative accordée à l'objectif f_i relativement aux autres, $\bar{f}_i$ est une version normalisée de la fonction f_i, pour ramener les f_i à la même échelle, définie grâce au concept de *distance relative*. Si x, y et z sont trois nombres réels, tels que : $y < x$ et $y < z$, la *distance entre* x et y, *relativement* à z, est le rapport : $d_z(x,y) = \frac{x-y}{z-y}$. $\bar{f}_i$ est défini par l'expression : $\bar{f}_i = d_{f_i^0}(f_i, g_i)$, où g_i est une borne inférieure non atteignable par le critère f_i ($f_i > g_i$) et f_i^0 est la valeur de la fonction objectif au point initial (ce dernier peut être soit défini par l'utilisateur, soit issu d'une optimisation précédente, soit engendré aléatoirement).

2.3.2. Approche d'optimisation

Dans toute la suite de ce chapitre, nous nous intéresserons à l'étape d'optimisation approchée de l'approche multicritère présentée dans la section précédente. Cette étape revient à résoudre le programme mathématique bi-objectif (voir section 2.1.2) :

$$\left(\mathrm{P}_1^{(k)}\right)\begin{cases} Min\, f(x), S(x) \\ x \in \mathrm{X_k} \\ f_j(x) \le b_j,\ j \in \mathrm{J} \end{cases}$$

où $f = \sum_{i \in \mathrm{I}} w_i \bar{f}_i$.

Les contraintes ($f_j \le b_j, j \in \mathrm{J}$) dans $\left(\mathrm{P}_1^{(\mathrm{k})}\right)$ ont été prises en compte à l'aide d'une méthode de pénalité, qui consiste à ajouter un second terme à la fonction objectif, pour pénaliser les solutions qui ne respectent pas ces contraintes.

Si les objectifs de l'utilisateur sont d'optimiser les critères $f_i, i \in I$ en imposant des bornes supérieures sur $f_j, j \in J$, le programme mathématique à résoudre s'écrira :

$$\left(\mathrm{P}_2^{(k)}\right)\begin{cases} Min\, f(x), S(x) \\ \\ x \in \mathrm{X}_k \end{cases},$$

où $f = \sum_{i \in I}\left[w_i\, d_{f_i^0}(f_i, g_i)\right] + C \sum_{j \in J} \max\left(d_{f_j^0}(f_j, b_j), 0\right)$, b_j est le second membre de la contrainte ($f_j \le b_j$), et C un coefficient de pénalité associé aux contraintes (nombre réel suffisamment grand).

Tout au long de ce chapitre, nous présenterons plusieurs méthodes pour résoudre le programme $\left(\mathrm{P}_2^{(k)}\right)$, basées sur l'utilisation de métaheuristiques. Le programme $\left(P_2^{(k)}\right)$ poursuit deux objectifs : minimiser la fonction *f* (problématique de réglage) et minimiser la fonction S (problématique de sélection). L'idée que nous proposons est d'optimiser seulement le réglage, en orientant l'exploration de l'espace de recherche vers des solutions ne contenant pas trop de paramètres sélectionnés, en pénalisant la fonction objectif, en éliminant des paramètres jugés inutiles, en se restreignant à des solutions avec peu de paramètres sélectionnés…. Toutes ces stratégies de sélection seront décrites dans la section suivante.

Les métaheuristiques utilisent en général des opérateurs de variation pour construire de nouvelles solutions à partir des anciennes. Nous utiliserons les deux opérateurs de variation suivants :

- *la mutation***,** qui est un opérateur défini sur l'espace des solutions : $\eta : \mathrm{X} \to \mathrm{X}$, tel que : $\eta(x)$ est une solution construite en effectuant des changements sur les coordonnées de *x*.
- *La recombinaison*, qui est un opérateur binaire : $\nu : \mathrm{X} \times \mathrm{X} \to \mathrm{X}$, tel que $\nu(x, y)$ est une solution construite en combinant *x* et *y*.

L'application d'un opérateur de variation (mutation ou recombinaison) à un point $x = (x_1, x_2, ..., x_n) \in [0,1]^n$ peut donner un point $y = (y_1, y_2, ..., y_n)$ à l'extérieur de l'espace $[0,1]^n$, c'est-à-dire que les valeurs de certaines variables y_i se trouvent en dehors de l'intervalle $[0,1]$. Dans ce cas, une méthode triviale pour ramener les y_i dans leurs intervalles de variation est la *méthode de rejet*, qui consiste à rejeter le point *y* et à répéter l'application de l'opérateur de variation jusqu'à ce que toutes les variables se retrouvent dans leurs intervalles respectifs. Cette méthode peut devenir très coûteuse lorsque l'on travaille près des bornes (car la plupart des points engendrés grâce à l'opérateur de variation peuvent se trouver en dehors de leurs domaines de variation).

Une autre méthode consiste à transformer le point y en un autre point z appartenant à $[0,1]^n$, en effectuant une projection sur les bornes (*« clipping method »)* :

$$z_i = \begin{cases} 1 & \text{si } y_i > 1 \\ 0 & \text{si } y_i < 0 \\ y_i & \text{sinon} \end{cases},$$

Comme la précédente, cette dernière méthode peut aussi devenir très coûteuse près des bornes, car les points engendrés peuvent être tous identiques. Nous avons donc opté pour une autre technique, appelée « *bouncing method »* qui consiste à effectuer une transformation par symétrie relativement aux bornes :

$$z_i = \begin{cases} 2 - y_i & \text{si } y_i > 1 \\ - y_i & \text{si } y_i < 0 \\ y_i & \text{sinon} \end{cases}.$$

2.4. Stratégies de sélection de paramètres

Nous avons vu dans ce qui précède que notre problème se ramène à l'optimisation de deux objectifs (le choix et le réglage des paramètres). Une technique bien connue en optimisation multicritère consiste à optimiser un critère et introduire les autres dans les contraintes. Cette technique, appelée *méthode ε-contrainte* [Miettinen, 1999] ou *méthode du compromis* [Collette et Siarry, 2002], a été introduite dans [Haimes et al., 1971]. L'idée présentée ici est assez proche, avec la différence que les contraintes prennent une forme plus générale et ne sont pas restreintes à imposer une borne maximum aux objectifs.

Nous proposons d'utiliser des métaheuristiques pour optimiser le réglage des paramètres et incorporer le second objectif (choix de paramètres) dans les opérateurs permettant d'engendrer ou de sélectionner des solutions. En d'autres termes, à partir du programme bi-objectif $\left(P_2^k\right)$ (voir paragraphe 2.2) on se ramène à la résolution du problème suivant :

$$\left(P_3^{(k)}\right) \begin{cases} Min\, f(x) \\ \\ x \in \mathrm{X}'_k \end{cases}$$

où X'_k est un sous-ensemble de X_k obtenu en éliminant les solutions avec « trop » de paramètres sélectionnés.

Notre approche s'inspire aussi des techniques utilisées pour la résolution de problèmes similaires rencontrés :

- *en statistique :* sélection de variables en régression linéaire multiple [Miller, 1990],
- *en apprentissage :* sélection d'attributs Kohavi et John, 1997.

Plusieurs stratégies de sélection de paramètres peuvent être utilisées :

- *Stratégie d'initialisation (SI) :* une étape essentielle dans tout algorithme basé sur les métaheuristiques est l'initialisation de la recherche. Des exemples sont la construction de la population initiale dans un algorithme évolutionnaire, ou le choix de la solution initiale pour démarrer une recherche locale. L'utilisation d'une stratégie (SI) consiste à partir d'une solution donnée (en général la solution nulle) et appliquer la mutation uniforme (mutation qui consiste à changer un paramètre en lui affectant une valeur choisie aléatoirement entre ses bornes) à un nombre r de paramètres choisis aléatoirement et uniformément (le nombre r est une constante fixée par l'utilisateur). Si la solution initiale est nulle, les solutions générées seront ainsi constituées de r paramètres non nuls. L'intérêt de cette technique d'initialisation est de détecter les groupes de paramètres intéressants. Une variante de cette stratégie d'initialisation, qui s'applique au cas où certains paramètres sont déjà sélectionnés dans la solution initiale, consiste à appliquer la mutation uniforme uniquement aux paramètres sélectionnés, ce qui permet de restreindre la recherche à ces paramètres et de maintenir le même nombre de paramètres sélectionnés. Cette dernière stratégie sera notée (SIR).

- *Stratégie de comparaison (SC) :* pour comparer deux solutions, on considère les valeurs de la fonction objectif f ; si leurs valeurs sont égales, les valeurs de la fonction S sont considérées (l'individu avec le plus faible nombre de paramètres est considéré comme meilleur). En d'autres termes, la comparaison entre les individus est basée sur une méthode lexicographique, avec une priorité plus faible pour l'objectif de sélection par rapport au réglage.

- *Stratégie de recombinaison (SR) :* l'étape de recombinaison est réalisée grâce à l'opérateur de recombinaison discrète, qui consiste à construire une nouvelle solution z à partir de deux[16] solutions x et y, en choisissant les valeurs des paramètres z_i avec une probabilité uniforme sur $\{x_i, y_i\}$. Cet opérateur possède deux propriétés intéressantes :

  - seuls les paramètres sélectionnés dans x ou dans y peuvent être sélectionnés dans z,

  - un paramètre sélectionné simultanément dans x et y est nécessairement sélectionné dans z.

  L'idée intuitive derrière l'utilisation de cet opérateur de recombinaison consiste à considérer que les paramètres intéressants sont ceux appartenant à plusieurs bonnes solutions. Cette hypothèse appelée *« commonality hypothesis »* est un principe général utilisé pour la construction d'opérateurs de recombinaison, en particulier pour la sélection d'attribut en apprentissage automatique [Chen, 1999].

  Un inconvénient d'une telle méthode de recombinaison est que, dans la solution construite, le nombre de paramètres peut être trop élevé ou trop faible. Pour y remédier, on peut imposer une borne supérieure $\overline{S}$ et une borne inférieure $\underline{S}$ au nombre de paramètres sélectionnés dans la solution construite avec l'opérateur de recombinaison. Si cette contrainte n'est pas vérifiée,

[16] La recombinaison discrète peut aussi s'appliquer à plusieurs individus.

l'opérateur de recombinaison est appliqué à nouveau et les contraintes sont vérifiées. Afin de ne pas alourdir les calculs (dans le cas où les contraintes sont difficiles à respecter), l'opérateur est appliqué au plus *e* fois (*e* étant un nombre entier positif donné). Au terme de ces *e* applications, un terme de pénalité (constante réelle de valeur très élevée) est ajouté à la valeur de la fonction objectif *f* en ce point.

- *Stratégie de mutation (SM) :* l'opérateur de mutation est appliqué en se restreignant aux paramètres sélectionnés (c'est-à-dire à valeurs non nulles), de manière à ce que la nouvelle solution ne contienne pas plus de paramètres sélectionnés que l'ancienne. Ceci permet d'intensifier la recherche dans le sous-espace des paramètres sélectionnés.

  L'inconvénient d'une telle stratégie est qu'elle n'exploite pas les paramètres non sélectionnés, alors que certains d'entre eux peuvent permettre d'améliorer la solution courante. L'idée est alors de muter un paramètre non sélectionné avec une certaine probabilité. La distribution que nous utiliserons par la suite est donnée par :

$$P = \begin{cases} 0 \text{ si } S(x) \geq \overline{S} \\ \rho \text{ si } 0 < S(x) < \overline{S} \\ 1 \text{ si } S(x) = 0 \end{cases},$$

  où les deux constantes ρ ($0 < \rho < 1$) et $\overline{S}$ (nombre maximum de paramètres sélectionnés) sont spécifiées par l'utilisateur.

- *Élimination aléatoire en arrière (SE)* : cette stratégie consiste à parcourir les paramètres sélectionnés aléatoirement et à éliminer ceux qui sont inutiles (c'est-à-dire ceux pour lesquels la solution courante n'est pas dégradée s'ils sont éliminés). Cette stratégie est inspirée de l'algorithme d'*« élimination aléatoire en arrière »* rencontrée en statistique (voir [Miller, 1990]) et en apprentissage automatique (voir).

- *Sélection aléatoire en avant (SS)* : comme la précédente, cette stratégie est similaire à l'algorithme de « sélection aléatoire en avant » rencontré en statistique et en apprentissage automatique. Elle peut aussi être vue comme une forme de descente aléatoire (voir chapitre 1 pour une description de ce type de méthode). Elle consiste à appliquer une mutation particulière η_s de façon répétée à une solution. Après chaque mutation, si la nouvelle solution n'est pas meilleure que la solution courante, elle est rejetée. Dans le cas contraire, la nouvelle solution remplace la solution courante. La mutation η_s opère en appliquant une mutation η (quelconque) à un paramètre choisi de façon aléatoire et uniforme, soit parmi tous les paramètres (version large), soit parmi les paramètres non sélectionnés seulement (version stricte).

- *Sélection et élimination aléatoire (SSE)* : cette stratégie est une amélioration de la précédente (SS), en lui ajoutant la possibilité d'éliminer un paramètre, s'il est considéré comme inutile. Au lieu d'utiliser uniquement la mutation η_s, on peut éliminer à chaque itération un paramètre avec une probabilité qui dépend

du nombre de paramètres sélectionnés dans la solution courante. La probabilité d'éliminer un paramètre est définie par :

$$P(x)=\begin{cases}0 & \text{si } S(x)\le\theta\\ \exp\left(\dfrac{-\alpha}{S(x)-\theta}\right) & \text{sinon}\end{cases},$$

où x est la solution courante, θ est un entier positif et α une constante réelle positive.

Si le nombre de paramètres sélectionnés de x est inférieur à θ, seule la mutation η_s est appliquée, ce qui tend à accroître le nombre de paramètres sélectionnés dans x. Dès que ce nombre dépasse θ , la probabilité d'éliminer un paramètre est positive et croît avec la différence ($S(x)-\theta$), ce qui permet de ne pas avoir un nombre de paramètres sélectionnés beaucoup plus élevé que θ . En effet, au fur et à mesure qu'on s'éloigne de θ , la probabilité d'éliminer des paramètres s'accroît (la constante α permet de contrôler la vitesse d'accroissement, et $S(x)$ tend à revenir vers les valeurs proches de θ).

2.5. Méthodes d'optimisation

Nous présenterons ici plusieurs approches pour la résolution du programme $\left(P_3^{(k)}\right)$ (voir la partie précédente) :

$$\left(P_3^{(k)}\right)\begin{cases}Min\, f(x)\\ \\ x\in X'_k\end{cases}$$

Pour illustrer les stratégies présentées dans la partie précédente, nous avons choisi d'implémenter quelques métaheuristiques parmi les plus simples et les plus faciles à mettre en œuvre : les algorithmes basés sur la descente aléatoire et les algorithmes évolutionnaires.

2.5.1. Algorithmes basés sur la descente aléatoire

La plupart des métaheuristiques à base de solution unique sont des améliorations de la méthode de descente aléatoire. Les plus simples sont des variantes de la descente aléatoire répétée, qui consiste à faire une descente aléatoire à partir de plusieurs points choisis de façon aléatoire dans l'espace de recherche, et la méthode du Kangourou, qui sera présentée plus loin.

2.5.1.1. Méthode de Descente Aléatoire répétée

Dans une descente aléatoire répétée, un seul point est tiré de façon aléatoire, puis utilisé pour démarrer une optimisation locale. Un élément notable de cette procédure est que le point initial ainsi obtenu peut être très éloigné de l'optimum global. Une amélioration possible consiste à tirer aléatoirement plusieurs points, et à choisir le

meilleur pour démarrer l'optimisation locale, en évitant ainsi d'intensifier la recherche dans les régions peu prometteuses. La méthode décrite dans l'algorithme 2.1 utilise une population de points initiaux, qu'elle met à jour avec de nouveaux points tirés de manière aléatoire après chaque optimisation locale. Le meilleur point de cette population est utilisé pour démarrer une descente aléatoire, après qu'il ait été éliminé de la population.

1. Construire une population P de μ points initiaux ;
2. Sélectionner la meilleure solution x_0 de P et éliminer x_0 de P ;
3. Initialiser la solution courante : $x \leftarrow x_0$;
4. Initialiser la meilleure solution rencontrée : $x^* \leftarrow x_0$;
5. $x_1 \leftarrow$ DescenteAléatoire(x) ;
6. Si $f(x_1) < f(x^*)$, Alors $x^* \leftarrow x_1$;
7. Insérer λ nouvelles solutions obtenues par échantillonnage aléatoire, ensuite éliminer les $(\lambda - 1)$ plus mauvaises solutions de P ;
8. Si le critère d'arrêt est atteint, Alors fin de l'algorithme,
   Sinon aller en 5.

Algorithme 2.1 - *Descente aléatoire répétée à base de population de points initiaux.*

L'algorithme ci-dessus peut être vu comme une répétition de deux étapes : une étape d'exploration, qui consiste à construire des points par échantillonnage aléatoire dans l'espace de recherche, et une étape d'exploitation, dans laquelle on démarre une descente à partir du meilleur point obtenu au cours des explorations précédentes. Dans l'étape exploratoire, nous utiliserons la stratégie d'initialisation (SI), ce qui permettra de sélectionner un sous-ensemble de paramètres intéressants. On utilisera ensuite la stratégie (SM) avec la mutation de descente, pour intensifier la recherche dans l'espace des paramètres sélectionnés lors de l'exploration.

2.5.1.2. Méthode du Kangourou

La méthode de descente du Kangourou a été proposée par [Fleury, 1993]. Elle présente l'avantage de ne pas perdre l'information relative aux optima locaux rencontrés. Après une descente aléatoire avec une mutation η_1, si la valeur de la fonction objectif n'a pas changé depuis A itérations, plusieurs sauts aléatoires consécutifs sont effectués en utilisant une mutation η_2 (voir algorithmes 2.2, 2.3 et 2.4). La mutation η_2 n'est pas nécessairement la même que η_1, mais doit respecter la propriété d'accessibilité, c'est-à-dire que pour tout couple de points (x, y) de l'espace des paramètres, il doit être possible d'atteindre y à partir de x, en utilisant une suite finie de mutations de type η_2. Cette propriété est suffisante pour garantir la convergence

asymptotique de l'algorithme [Fleury, 1993]. La preuve de cette convergence repose sur les propriétés théoriques liées aux chaînes de Markov.

1. Initialiser la solution courante : $x \leftarrow x_0$;
2. Initialiser la meilleure solution rencontrée : $x^* \leftarrow x_0$;
3. Initialiser le compteur de stationnement : C $\leftarrow$ 1 ;
4. Si $C < A$ alors exécuter la procédure de descente : $x \leftarrow$ descente(x, C) ;
   Sinon exécuter la procédure de saut : $x \leftarrow$ saut(x) ;
5. Si x est meilleure que x^* alors $x^* \leftarrow x$;
6. Si le critère d'arrêt est atteint alors aller en 4,
   Sinon fin de l'algorithme.

Algorithme 2.2 - *Méthode du kangourou.*

1. Appliquer la mutation η_1 à la solution courante : $x_1 \leftarrow \eta_1(x)$;
2. Si $f(x_1) > f(x)$ alors aller en 5 ;
3. Si $f(x_1) < f(x)$ alors $C \leftarrow 0$;
4. $x \leftarrow x_1$;
5. $C \leftarrow C+1$;

Algorithme 2.3 - *Procédure de descente.*

Répéter *ns* fois :

1. Appliquer la mutation η_2 à la solution courante : $x_1 \leftarrow \eta_2(x)$;
2. Si $f(x_1) = f(x)$ alors aller en 5 ;
3. Si $f(x_1) < f(x^*)$ alors mettre à jour la meilleure solution rencontrée : $x^* \leftarrow x_1$;
4. Réinitialiser le compteur de stationnement : $C \leftarrow 0$;
5. Mettre à jour la solution courante : $x \leftarrow x_1$;
6. Incrémenter le compteur de stationnement : $C \leftarrow C+1$;

Algorithme 2.4 - *Procédure de saut.*

Les mutations η_1 et η_2 ont été choisies comme suit :

- η_1 : *mutation uniforme locale*. $\eta_1(x_i) = x_i + (2 \cdot \gamma - 1) \cdot \rho$, où γ est obtenu à partir d'une distribution uniforme sur $[0,1]$ et ρ est un nombre réel ($0 < \rho < 1$), souvent appelé *taille maximum du pas*. Cette mutation peut s'interpréter comme un déplacement vers un point choisi dans un N-cube centré en x et de côté 2ρ.

- η_2 : *mutation uniforme globale*. $\eta(x_i) = \gamma$, où γ est obtenu à partir d'une distribution uniforme sur $[0,1]$. La mutation η_2 s'interprète comme un déplacement aléatoire dans le N-cube $[0,1]^N$.

La mutation η_2 vérifie bien la propriété d'accessibilité, puisqu'à partir d'un point quelconque de l'espace de recherche $[0,1]^N$, il est possible d'atteindre tout autre point de cet espace. Les deux mutations η_1 et η_2 sont utilisées avec des objectifs différents. η_1 permet de faire un déplacement local (c'est-à-dire, vers un point très proche de la solution courante), alors que η_2 est utilisée pour effectuer un saut vers un autre *bassin d'attraction*, pour sortir d'un optimum local.

La figure 2.3 présente les stratégies de sélection de paramètres utilisées avec l'algorithme du Kangourou. Pour intensifier la recherche dans l'espace des paramètres sélectionnés, tout en donnant la possibilité aux autres d'être sélectionnés eux aussi, une stratégie de type (SM) est utilisée avec la mutation de descente η_1. Avec une telle stratégie, seuls les paramètres ayant permis d'améliorer la fonction objectif à une itération donnée peuvent être ajoutés à l'ensemble des paramètres sélectionnés. A la fin de chaque descente, et avant d'effectuer un saut, une stratégie d'élimination aléatoire en arrière (SE) est appliquée à la solution courante, afin d'éliminer les paramètres inutiles. Ensuite, les sauts sont effectués en utilisant une stratégie d'initialisation restreinte (SIR) avec la mutation de saut η_2, ce qui permet d'éviter l'augmentation du nombre de paramètres durant les sauts. La stratégie (SC) est utilisée à chaque comparaison entre solutions.

2.5.2. Algorithmes évolutionnaires

Le principe de base d'un algorithme évolutionnaire est le suivant : étant donnée une population d'individus, la pression de l'environnement entraîne une sélection naturelle des individus les plus adaptés, ce qui améliore l'adaptation des individus de la population. Si l'on se trouve devant une fonction objectif à maximiser, l'idée est de construire un ensemble de solutions candidates avec un tirage aléatoire, de les considérer comme des individus et d'utiliser la fonction objectif pour mesurer leur adaptation (l'adaptation d'un candidat croît avec la valeur de la fonction objectif). En se basant sur cette adaptation, certains candidats sont choisis (ce choix est fait de manière à simuler la sélection naturelle, c'est-à-dire à favoriser les candidats les plus adaptés), pour engendrer de nouveaux candidats afin de former la génération suivante. Ces derniers candidats sont obtenus en appliquant des opérateurs de recombinaison et/ou de mutation. La recombinaison est un opérateur qui s'applique à deux ou plus candidats sélectionnés (les *parents*) pour engendrer un ou plusieurs nouveaux candidats (les *enfants*), alors que la mutation s'applique à un candidat et donne naissance à un seul nouveau candidat. L'application de recombinaisons et mutations a pour résultat un

ensemble de nouveaux candidats (la *progéniture*) qui sont en concurrence avec les anciens pour une place dans la génération suivante, concurrence basée sur leur adaptation (et peut être aussi sur leur âge). Le processus global est répété jusqu'à ce qu'un candidat avec une qualité suffisante (une solution) soit trouvé ou qu'une limite de temps de calcul ait été atteinte.

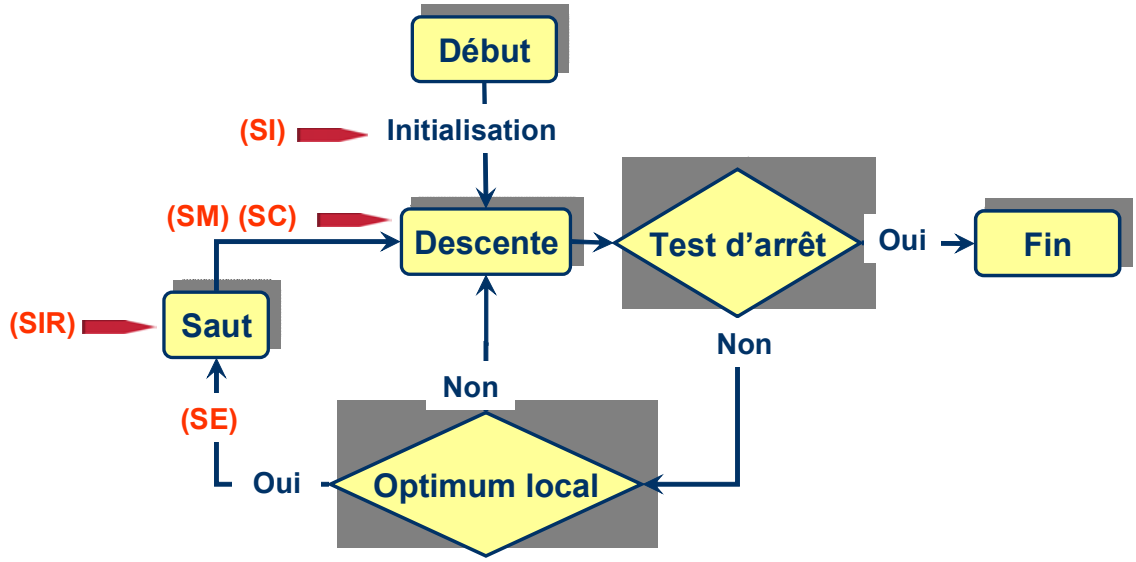


Figure 2.3 - *Utilisation des stratégies de sélection de paramètres au sein de l'algorithme du Kangourou.*

Le schéma général d'un algorithme évolutionnaire est donné dans l'algorithme 2.5 [Eiben et Smith, 2003].

En se basant sur ce schéma général, nous avons proposé la stratégie de sélection de paramètres suivante (voir figure 2.4) :

- une stratégie d'initialisation (SI) est utilisée pour la construction de la population initiale,
- une stratégie de recombinaison (SR) est appliquée à l'étape de recombinaison (ce qui implique que c'est l'opérateur de recombinaison discrète qui sera utilisé),
- une stratégie de mutation (SM) est appliquée à l'étape de mutation,
- la stratégie de comparaison (SC) est utilisée à chaque fois qu'on fait une sélection d'individus (pour être recombinés, ou pour les sélectionner pour la nouvelle génération).

Plusieurs algorithmes évolutionnaires ont été proposés dans la littérature pour optimiser des fonctions à variables continues. Afin de valider l'approche présentée précédemment, nous avons choisi trois méthodes, parmi les plus simples à implémenter :

- une version simple d'algorithme génétique à codage réel. Le choix du codage réel par rapport au codage classique basé sur les chaînes de bits est lié à plusieurs résultats récents, qui montrent la supériorité de ce type de codage pour l'optimisation continue.
- un algorithme basé sur les stratégies d'évolution avec mutation auto-adaptative, qui sont d'une grande efficacité dans l'optimisation continue,
- un algorithme génétique reproducteur.

Initialiser la population avec des solutions candidates obtenues par tirage aléatoire ;

Évaluer chaque candidat ;

Tant que le *critère d'arrêt* n'est pas atteint faire

1. *Sélectionner* les parents ;
2. *Recombiner* les couples de parents ;
3. *Muter* la progéniture ;
4. *Évaluer* les nouveaux candidats ;
5. *Sélectionner* des individus pour la génération suivante ;

Fait

Algorithme 2.5 - *Schéma général d'un algorithme évolutionnaire.*

2.5.2.1. Algorithme génétique à codage réel

Dans sa forme classique, un algorithme génétique est appliqué à un problème d'optimisation de paramètres réels en utilisant soit un codage binaire standard, soit un codage de Gray. Les chaînes de bits associées aux différents paramètres sont concaténées pour former une seule chaîne de bits (ou *chromosome*) représentant le vecteur de paramètres. Cette représentation binaire des solutions s'appuie sur des résultats théoriques, qui suggèrent qu'une représentation binaire est plus performante qu'une représentation basée sur un alphabet plus large. Cependant, des études plus récentes montrent la supériorité du codage réel (chaîne de nombres réels) sur le codage binaire pour optimiser des paramètres réels [Davis, 1991] [Wright, 1991] [Janikow et Michalewicz, 1991] [Michalewicz, 1994] [Surry et Radcliffe, 1996] [Ono et Kobayashi, 1997] [Ono et al. 1999].

Plusieurs études théoriques ont aussi été menées sur les algorithmes génétiques à codage réel, principalement sur l'analyse de certains opérateurs génétiques adaptés aux codages réels [Goldberg, 1991] [Grossman et Davidor, 1992] [Eshelman et Schaffer, 1993] [Qi et Palmieri 1994a, 1994b] [Kita et Yamamura, 1999] [Higuchi et al., 2000].

L'algorithme 2.7 décrit les grandes lignes d'une version simple d'algorithme génétique à codage réel.

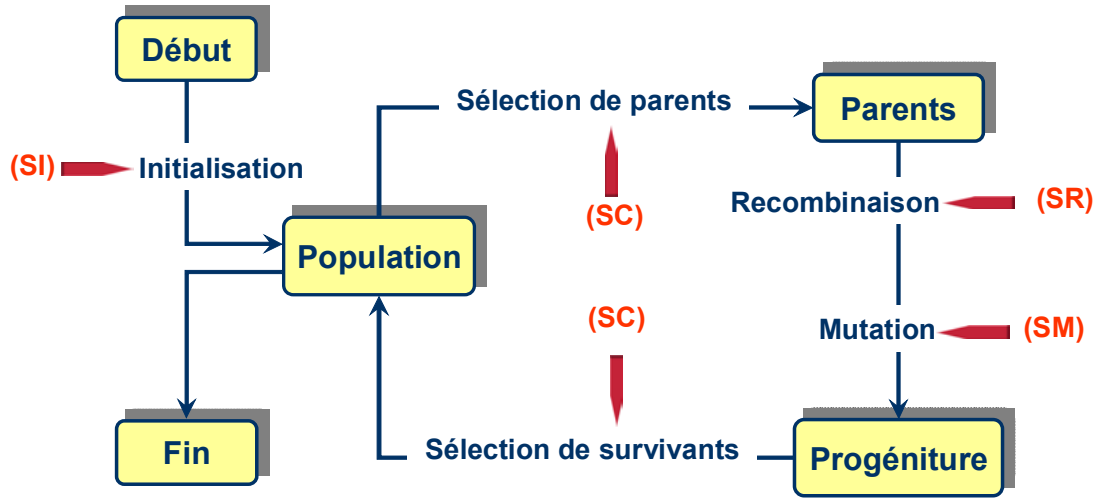


Figure 2.4 - *Utilisation des stratégies de sélection de paramètres au sein d'un algorithme évolutionnaire.*

1. Construire une population initiale $P^{(0)}$ de μ individus ;
2. Initialiser le compteur de générations : $t \leftarrow 0$;
3. Évaluer les adaptations des individus de la génération t : $P^{(t)}$;
4. Construire une population $P_1^{(t)}$ de μ individus en appliquant l'opérateur de recombinaison à des couples sélectionnés sur $P^{(t)}$ avec l'opérateur de sélection ;
5. Construire une population $P_2^{(t)}$ en appliquant l'opérateur de mutation aux individus de $P_1^{(t)}$;
6. Évaluer les adaptations des individus de $P_2^{(t)}$;
7. La nouvelle génération $P^{(t+1)}$ est obtenue en sélectionnant les μ meilleurs individus dans $P_2^{(t)} \cup \{x^{(t)}\}$, où $x^{(t)}$ est le meilleur individu dans $P^{(t)}$;
8. $t \leftarrow t + 1$;
9. Si le critère d'arrêt est atteint, alors aller en 3,
   Sinon fin de l'algorithme.

Algorithme 2.7 – *Version simple d'algorithme génétique à codage réel.*

Les *opérateurs génétiques* utilisés sont :

- la mutation gaussienne, qui est l'opérateur classique pour les codages réels [Eiben et Smith, 2003] : $\eta(x_i) = x_i + \sigma \cdot \gamma$, où γ est généré à partir d'une loi normale centrée et d'écart type σ.
- La recombinaison discrète, pour ses propriétés intéressantes dans la sélection de paramètres (stratégie de recombinaison (SR)).
- Pour éviter les problèmes d'échelle, de convergence prématurée et de pression sélective des schémas traditionnels basés sur les valeurs de la fonction objectif (comme la méthode de la roulette de casino), la sélection de solutions pour la recombinaison est obtenue grâce à l'opérateur de « rangement linéaire » [Whitley, 1989]. La population est triée selon les valeurs de la fonction objectif (du plus mauvais individu au meilleur). Une adaptation est ensuite calculée pour chaque individu en se basant sur son rang dans la liste. Si *pos* est la position d'un individu dans la population, son adaptation est calculée comme suit : $2 - SP + 2\,(SP-1)\,(pos-1)/(\mu-1)$, où $SP \in [1.0, 2.0]$ (la valeur $SP = 1.5$ a été utilisée dans les expérimentations numériques). La méthode classique de la roulette de casino est ensuite appliquée à la fonction d'adaptation précédente.

2.5.2.2. Stratégies d'évolution

Les algorithmes d'évolution sont typiquement utilisés pour l'optimisation de variables continues. Ils utilisent généralement des mutations auto-adaptatives : les écarts types des mutations gaussiennes font partie des individus et subissent également la mutation. Un individu est représenté sous la forme $(x_1, x_2, ..., x_n, \sigma_1, \sigma_2, ..., \sigma_{n_\sigma})$ (dans une forme plus générale, d'autres paramètres sont ajoutés à l'individu pour représenter l'interaction entre les écarts types).

Deux types de mutations gaussiennes sont généralement utilisés, selon que les écarts types soient identiques pour tous les x_i (nombre d'écarts types $n_\sigma = 1$) ou différents pour chaque x_i ($n_\sigma = n$) :

- **Cas $n_\sigma = 1$.** Dans ce cas, la même distribution est appliquée pour chacun des x_i, selon les formules suivantes :

  $\sigma' = \sigma \cdot e^{\tau \cdot \mathrm{N}(0,1)}$,

  $x_i' = x_i + \sigma' \cdot \mathrm{N}_i(0,1)$,

  et $\tau \propto n^{-1/2}$ (τ est proportionnelle à l'inverse de la racine carrée de *n*).

  $\mathrm{N}(0,1)$ et $\mathrm{N}_i(0,1)$, $i = 1, 2, ..., n$, désignent $n+1$ variables aléatoires appartenant à la loi normale centrée réduite.

- **Cas $n_\sigma = n$.** L'idée est ici d'utiliser des écarts types différents pour les différentes dimensions $i = 1, 2, ..., n$. Les formules appliquées dans ce cas sont :

  $\sigma_i' = \sigma_i \cdot e^{\tau' \cdot \mathrm{N}(0,1) + \tau \cdot \mathrm{N}_i(0,1)}$

$$x_i' = x_i + \sigma_i \cdot N_i(0,1)$$

$$\tau' \propto (2n)^{-1/2}$$

$$\tau \propto \left(2\sqrt{n}\right)^{-1/2}$$

Les valeurs de τ et τ' dans les formules ci-dessus sont recommandées dans [Schwefel, 1977].

1. Construire une population initiale $P^{(0)}$ de μ individus ;
2. Initialiser le compteur de générations : $t \leftarrow 0$;
3. Évaluer les adaptations des individus de $P^{(t)}$;
4. Construire une population $P_1^{(t)}$ de λ individus en appliquant l'opérateur de recombinaison à des couples sélectionnés selon une distribution de probabilité uniforme sur $P^{(t)}$;
5. Construire une population $P_2^{(t)}$ en appliquant l'opérateur de mutation aux individus de $P_1^{(t)}$;
6. Évaluer les adaptations des individus de $P_2^{(t)}$;
7. La nouvelle génération $P^{(t+1)}$ est définie en sélectionnant les μ meilleurs individus dans $P_2^{(t)}$ (resp. dans $P^{(t)} \cup P_2^{(t)}$);
8. $t \leftarrow t + 1$;
9. Si le critère d'arrêt est atteint, alors aller en 3,
   Sinon fin de l'algorithme.

Algorithme 2.8 - *Stratégie d'évolution (μ, λ) (resp. $\mu + \lambda$).*

L'étape de recombinaison est réalisée grâce à l'opérateur de recombinaison discrète, les parents étant choisis aléatoirement et uniformément dans la population courante, puis recombinés pour former un enfant. Cet opérateur est appliqué aux deux parties qui constituent la représentation d'un individu donné (variables et écarts types).

2.5.2.3. Algorithme génétique reproducteur (*Breeder Genetic Algorithm*)

Les algorithmes génétiques reproducteurs ont été conçus par Heinz Mühlenbein en Allemagne au début des années 90 [Mühlenbein et Schlierkamp, 1993 et 1994]. Ils se situent à mi-chemin entre les algorithmes génétiques et les stratégies d'évolution, dans la mesure où ils partagent avec chaque technique quelques idées de base. Ils ressemblent aux stratégies d'évolution par le fait qu'ils travaillent directement sur des variables réelles, les opérateurs de recombinaison de base n'étant pas très différents de ceux des stratégies d'évolution. Cependant, contrairement aux stratégies d'évolution, ils sont

principalement basés sur la recombinaison ; la mutation étant considérée comme un opérateur secondaire, à appliquer avec une probabilité faible.

1. Construire une population initiale $P^{(0)}$ de μ individus ;
2. Initialiser le compteur de générations : $t \leftarrow 0$;
3. Évaluer les adaptations des individus de $P^{(t)}$;
4. Construire $P_1^{(t)}$ en sélectionnant les τ % meilleurs individus de $P^{(0)}$;
5. Construire une population $P_2^{(t)}$ de $(\mu - q)$ individus en appliquant l'opérateur de recombinaison à des couples sélectionnés selon une distribution de probabilité uniforme sur $P_1^{(t)}$;
6. Construire une population $P_3^{(t)}$ en appliquant l'opérateur de mutation aux individus de $P_2^{(t)}$;
7. Évaluer les adaptations des individus de $P_3^{(t)}$;
8. La nouvelle génération $P^{(t+1)}$ est obtenu à partir de $P_3^{(t)}$ en insérant les q meilleurs éléments de $P^{(t)}$;
9. $t \leftarrow t + 1$;
10. Si le critère d'arrêt est atteint, alors aller en 3,
    Sinon fin de l'algorithme.

Algorithme 2.9 - *Algorithme génétique reproducteur.*

Les algorithmes génétiques reproducteurs, à la différence des algorithmes génétiques qui modélisent l'évolution naturelle, se basent sur une idée provenant du mécanisme de sélection utilisé dans la reproduction des animaux. Les éléments sélectionnés s'accouplent librement (l'accouplement d'un individu avec lui-même étant interdit), de manière à engendrer une nouvelle population, et les q meilleurs individus de l'ancienne population sont insérés dans la nouvelle (élitisme). La sélection des éléments qui vont s'accoupler obéit au modèle de troncature : seuls les τ% meilleurs individus de la population courante sont choisis (les valeurs typiques de τ sont entre 10 et 50). Ensuite, deux individus sont sélectionnés parmi eux de façon aléatoire et uniforme pour s'accoupler et engendrer un nouvel individu. Ceci est répété plusieurs fois, jusqu'à ce que l'on obtienne une nouvelle population de $\mu - q$ individus (voir algorithme 2.9 pour une description en pseudo-code).

Le schéma de sélection utilisé par les algorithmes génétiques reproducteurs possède les propriétés suivantes :

- il n'y a pas de tirages probabilistes (sélection déterministe),
- les éléments les plus mauvais ne survivent pas d'une génération à la suivante,

- les q meilleurs individus survivent toujours d'une génération à la suivante (sélection q-élitiste).

Pour un individu donné, l'application d'une mutation revient à choisir une variable de façon équiprobable parmi les *n*, puis à appliquer la transformation suivante :

$x_i' = x_i + \varepsilon_i \cdot \rho \cdot \delta$,

où ε_i est choisi de façon équiprobable dans $\{-1,1\}$, $\rho \in [0.1, 0.5]$ et le pas δ étant défini de la façon suivante :

$\delta = \sum_{i=0}^{k-1} \varphi_i 2^{-i}$, où φ_i suit une loi de $\text{Bernouilli}(1/k)$ et k est un entier positif.

La mutation ci-dessus est appelée *mutation discrète*. Signalons qu'il existe une deuxième mutation, appelée *mutation continue*, qui est obtenue avec la même formule que celle de la mutation discrète, mais avec $\delta = 2^{-k\beta}$, où β est tiré selon une loi uniforme sur $[0,1]$ [Belanche, 1999].

2.6. Conclusion

Tout au long de ce chapitre, nous avons présenté une approche de résolution pour la problématique de choix et de réglage de paramètres en entrée d'un logiciel d'ordonnancement. L'approche adoptée est une méthode multicritère interactive, dont l'étape de calcul est basée sur une métaheuristique. Deux types de métaheuristiques ont été adoptées : les méthodes de recherche locale et les méthodes évolutionnaires[17].

Les deux problèmes de choix et de réglage de paramètres sont résolus simultanément en introduisant des stratégies de sélection de paramètres dans les métaheuristiques. Toutes ces méthodes ont été implémentées en Java et regroupées au sein d'un prototype (*Ortems Optimizer)*, qui sera décrit dans le chapitre suivant. Plusieurs applications à des problèmes d'ordonnancement industriel sont aussi présentées.

[17] Notons ici que certains auteurs considèrent les méthodes évolutionnaires comme faisant partie des méthodes de recherche locale.

Chapitre 3

Application au paramétrage du logiciel d'ordonnancement industriel Ortems : expérimentations et résultats

3.1. Introduction

Ce chapitre est structuré autour de trois parties indépendantes. La première est une présentation d'un prototype destiné au paramétrage de logiciels d'ordonnancement industriels, dans lequel ont été implémentées plusieurs métaheuristiques exposées dans le chapitre précédent. Dans toute la suite du chapitre, ce prototype est appliqué au problème du paramétrage du logiciel d'ordonnancement *Ortems*. Celui-ci sera décrit dans la deuxième partie. La dernière partie sera consacrée à des expérimentations numériques visant à valider, illustrer et comparer les méthodes implémentées.

L'expérimentation de ce type d'algorithmes est difficile, en raison :

- de leur nature non déterministe, deux expériences identiques pouvant conduire à des résultats différents,
- de leur nature itérative, la solution étant d'autant meilleure que le temps de calcul est important,
- du coût élevé d'évaluation de la fonction objectif : chaque évaluation nécessite en effet de réaliser un ordonnancement,
- des problèmes liés à l'expérimentation d'un logiciel d'ordonnancement industriel, tels que les erreurs d'exécution, la complexité d'utilisation…

Plusieurs auteurs soulignent la difficulté de l'expérimentation numérique de méthodes non déterministes [Laguna et Adenso-Diaz, 2002] [Taillard, 2001]. Pour une

présentation de méthodologies pour la conduite de telles expériences, on pourra consulter par exemple [Barr et al., 1995 et 2001] ou [McGeoch et Moret, 1999].

3.2. Description de l'environnement *Ortems Optimizer*

3.2.1. Introduction

L'environnement *Ortems Optimizer* est un prototype basé sur les approches décrites dans le chapitre précédent, et permettant de régler les paramètres intervenant en entrée du logiciel d'ordonnancement *Ortems*, de manière à optimiser la performance des ses résultats. *Ortems Optimizer* utilise une approche d'optimisation en boîte noire (voir figure 3.1), dans laquelle le logiciel d'ordonnancement est considéré comme une boîte noire, avec des paramètres en entrée et des indicateurs de performance en sortie. La qualité d'une solution est évaluée après avoir effectué un ordonnancement du processus manufacturier considéré.

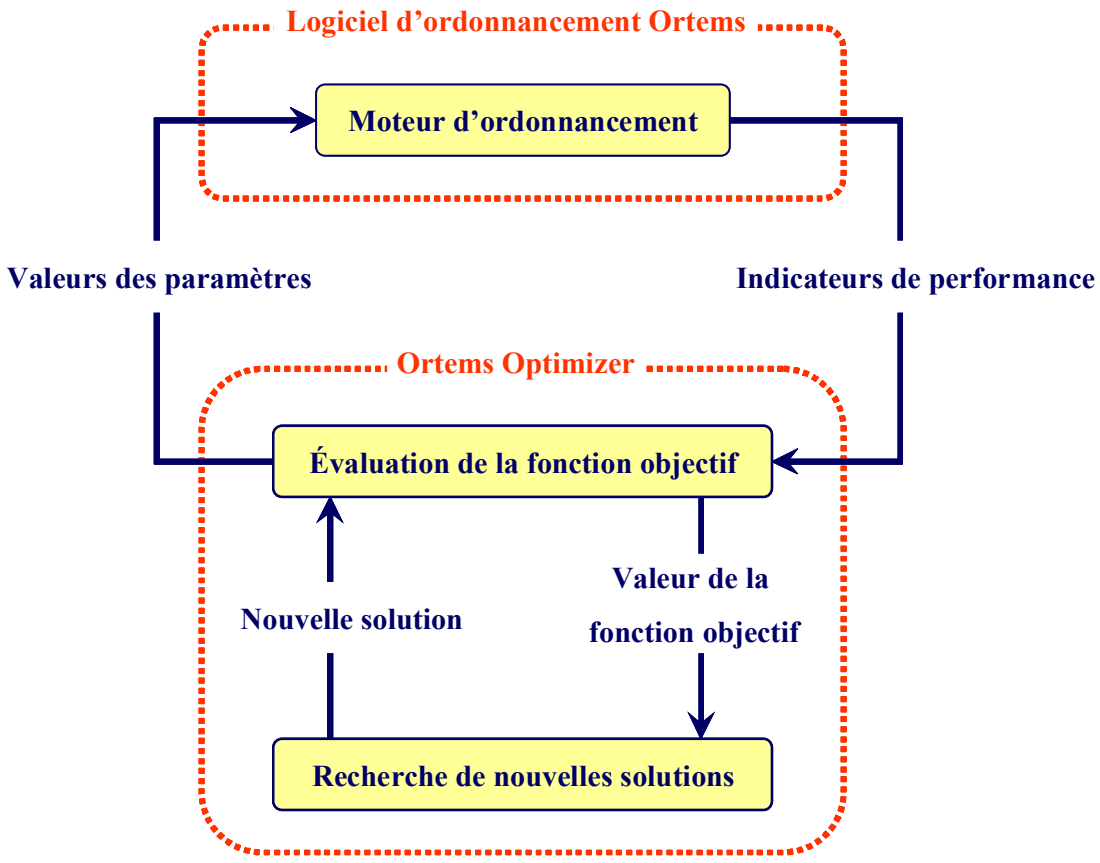


Figure 3.1 - *Optimisation en boîte noire.*

Chaque solution est évaluée en effectuant les quatre étapes suivantes :

- envoyer les valeurs des paramètres en entrée du moteur d'ordonnancement,
- réaliser un ordonnancement,
- récupérer les valeurs des indicateurs en sortie du moteur d'ordonnancement,
- calculer la fonction objectif.

Différentes méthodes d'optimisation sont utilisées pour la recherche de nouvelles solutions : descente aléatoire, recuit simulé, algorithme génétique…

3.2.2. Architecture

Le prototype *Ortems Optimizer* a été implémenté en « Java » et intégré dans l'architecture *e-SCM* de l'offre *Ortems*. La figure 3.2 montre l'architecture générale utilisée. Celle-ci est basée sur plusieurs modules :

- *Optimizer :* application programmée en langage « Java », basée sur les approches décrites dans le chapitre précédent et permettant de sélectionner, puis de régler, les paramètres utilisés en ordonnancement.
- *Serveur applicatif :* serveur implémenté en utilisant l'environnement « Delphi » et contenant le moteur d'ordonnancement *Ortems* (d'autres moteurs sont aussi présents). Le choix de l'environnement « Delphi » pour le développement de ce module est historique : les premiers programmes réalisés dans la société ont été écrits à la fin des années 80, en « Pascal ». Il a par la suite été plus simple de les porter sous l'environnement « Delphi » (dont le langage n'est autre qu'une version améliorée du « Pascal », basée sur la conception orientée objet).
- *Serveur e-SCM :* application programmée en langage « Java » permettant de gérer le partage des données du serveur applicatif entre plusieurs utilisateurs (pour un fonctionnement en mode collaboratif).

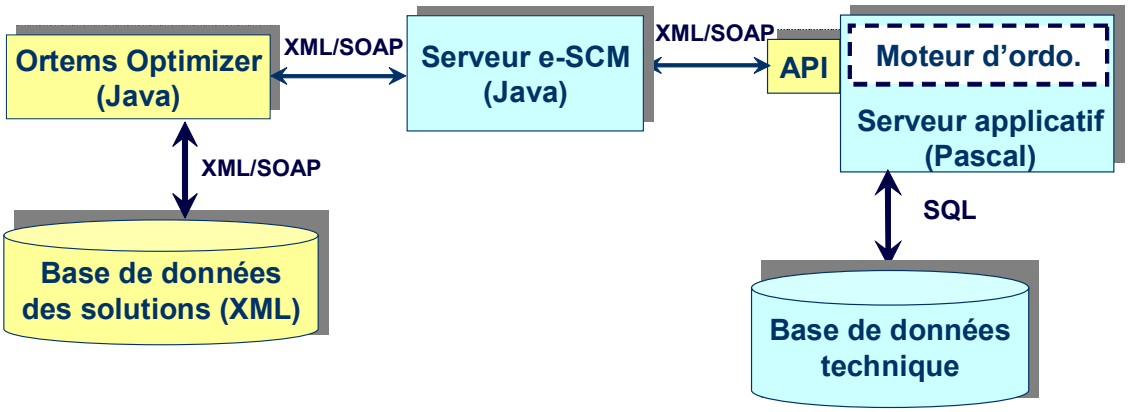


Figure 3.2 - *Architecture de l'application.*

Ces différents modules communiquent à l'aide de plusieurs techniques :

- *XML (eXtensible Markup Language) :* c'est un *langage de marquage* (c'est-à-dire un langage permettant de décrire des données textuelles en utilisant un codage basé sur des marqueurs) recommandé par la W3C (World Wide Web Consortium), pour faciliter l'échange et le partage de documents textuels structurés à travers Internet.
- *SOAP (Simple Object Access Protocol)* : c'est une technique permettant d'utiliser des applications invoquées à distance par Internet.
- *API (Application Programming Interface)* : c'est une interface (ou des conventions d'appels) permettant d'accéder à un ensemble de « routines ». L'une des principales tâches d'une API est la traduction des paramètres d'appel de ces « routines » d'un format (flux XML dans notre cas) à un autre (variables du langage de « Delphi »). Ces API ont été développées au sein du serveur applicatif.

Les données techniques du problème d'ordonnancement considéré (tâches, ordres de fabrication, machines, …) sont disponibles dans une base de données. Le langage de requêtes SQL est utilisé pour récupérer des données de la base.

Les informations relatives à une optimisation (méthode utilisée, configuration de la fonction objectif, meilleure solution obtenue, …) sont sauvegardées, à la demande de l'utilisateur, dans un fichier XML. Ces informations peuvent aussi être chargées à partir de ce même fichier.

3.2.3. Description fonctionnelle

Avant de lancer une optimisation, l'utilisateur doit choisir la fonction, qu'on appellera aussi *système*, à paramétrer dans le logiciel d'ordonnancement. Plusieurs systèmes peuvent être considérés, par exemple : ordonnancement total au plus tôt, ordonnancement total au plus tard, ordonnancement avec période figée au plus tôt, …. Chacun de ces systèmes est décrit dans un fichier XML en lui attribuant un nom, et en spécifiant les noms des API à appeler pour :

- récupérer des informations sur les paramètres utilisés en entrée du système, les indicateurs de performance…
- effectuer des actions sur le système : lancer un ordonnancement, valider une solution, mettre à jour les données techniques…

Nous désignerons par *réglage* l'ensemble des informations liées à une optimisation. Un réglage est composé de deux parties : configuration et résultats. La partie *configuration* contient les choix de l'utilisateur : sélection des paramètres à régler, définition de la fonction objectif, stratégie d'optimisation,… alors que, dans la partie *résultats,* on retrouve les résultats de la dernière optimisation : courbes d'évolution des indicateurs de performance, meilleure solution obtenue…

Dans le cas où les paramètres du système sont de nature ou de type différents, ils peuvent être rassemblés en *groupes*. Chaque groupe est constitué d'un ensemble de paramètres similaires. Si l'on considère notre problématique de réglage des poids associés aux règles de priorité utilisées en ordonnancement, on peut considérer deux

groupes de paramètres à valeurs continues : ceux associés aux règles d'ordonnancement et ceux associés aux règles de placement[18]. De plus, dans le moteur d'ordonnancement d'*Ortems*, il est possible d'associer des règles spécifiques à un îlot particulier, un îlot étant ici défini comme un ensemble de machines au fonctionnement similaire. Dans ce cas, les règles associées à cet îlot peuvent aussi être considérées comme un autre groupe de paramètres.

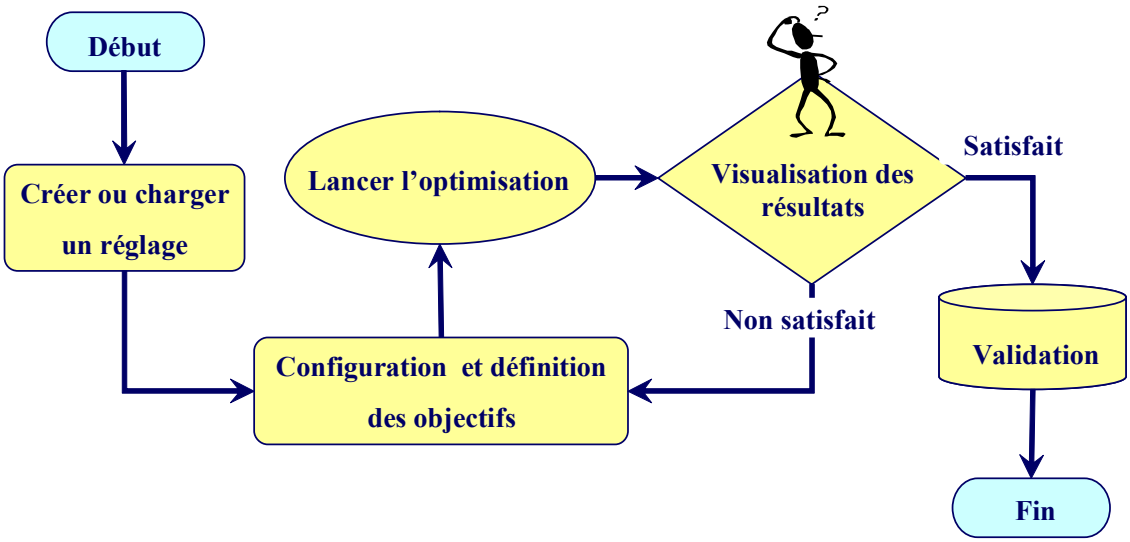


Figure 3.3 - *Étapes d'utilisation.*

La figure 3.3 montre les principales étapes d'utilisation de l'environnement *Ortems Optimizer*. Ces étapes sont conformes au schéma général décrit au chapitre 2, figure 2.2.

A la création d'un nouveau réglage, *Optimizer* récupère des informations sur les indicateurs de performance et les paramètres du système à optimiser, puis affiche plusieurs vues de configurations permettant à l'utilisateur d'introduire ses choix et ses préférences. Plusieurs graphiques et tables sont disponibles, permettant à l'utilisateur de suivre l'évolution des indicateurs de performance tout au long de l'optimisation. Si l'utilisateur juge les résultats satisfaisants, il peut valider la meilleure solution obtenue. L'opération de validation consiste à écrire les valeurs des paramètres dans la base de données techniques.

Le mode automatique est un fonctionnement particulier d'*Optimizer*, permettant un réglage périodique et automatique des paramètres de l'encours (voir figure 3.4). Le dernier état de l'encours est récupéré périodiquement, pour être mis à jour en fonction des opérations de suivi effectuées et des changements intervenus dans les données techniques.

[18] Voir la description de la méthode d'ordonnancement d'*Ortems* dans la partie 3.3.

3.2.4. Description technique

Le module *Ortems Optimizer* est constitué d'une centaine de classes programmées en langage Java et d'une trentaine de fichiers XML répartis principalement dans six modules interdépendants (voir figure 3.5) :

- *optimizer*, qui est le module principal,
- *system*, dont le rôle est l'interfaçage avec le système boîte noire à optimiser,
- *method*, qui s'occupe de la gestion des méthodes d'optimisation,
- *views*, qui gère la visualisation des courbes et tableaux de résultats, ainsi que les vues destinées à la saisie de la configuration des réglages,
- *menu*, dans lequel sont gérés les menus,
- *dialog*, permettant la gestion des boîtes de dialogue.

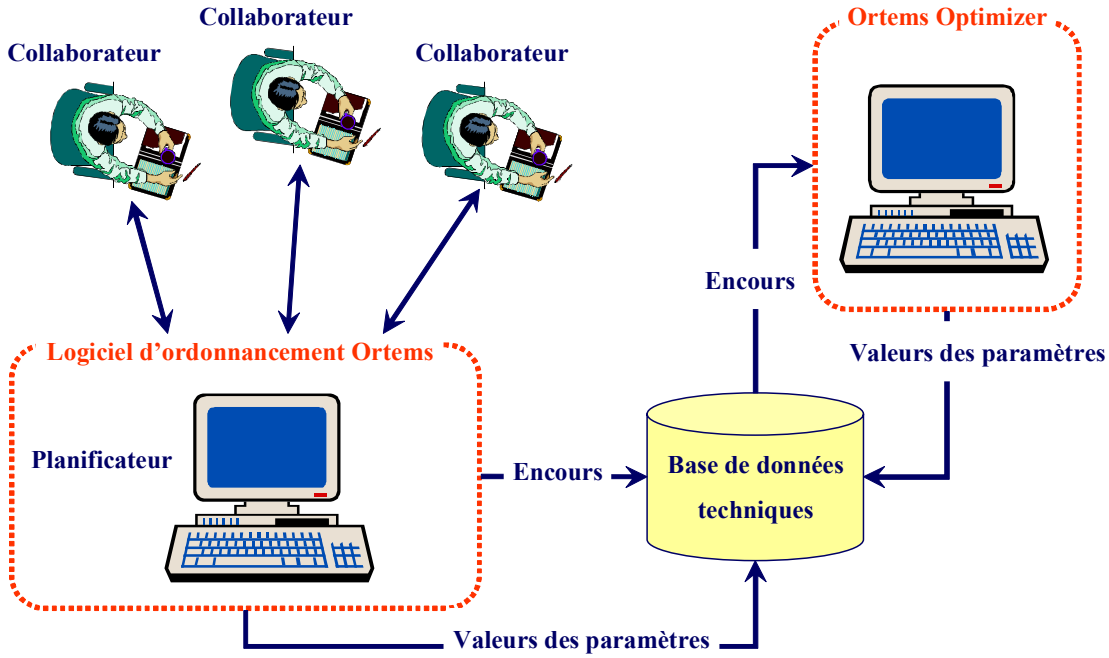


Figure 3.4 - *Mode automatique.*

L'implémentation du module *Ortems Optimizer* est basée sur une programmation orientée objet. Cette dernière est une méthode de programmation d'applications basée sur l'utilisation d'objets liés par des relations, par opposition à la programmation procédurale ou fonctionnelle, qui se base sur les fonctions de l'application à concevoir. L'une des caractéristiques de cette approche est la possibilité de concevoir de nouveaux objets à partir d'objets existants. Ainsi, il est possible de spécialiser un objet pour des besoins spécifiques.

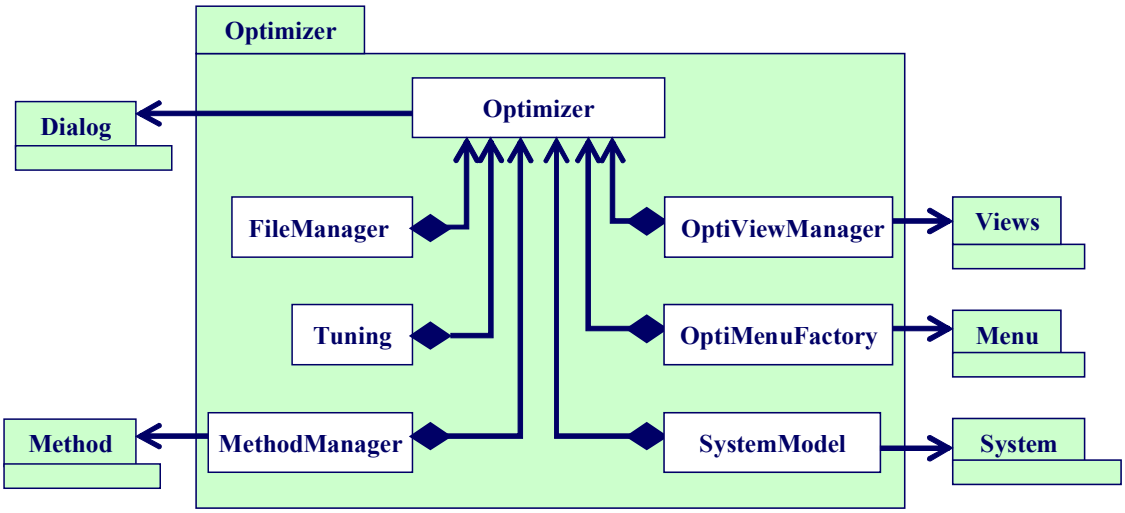


Figure 3.5 - *Composition du package Optimizer.*

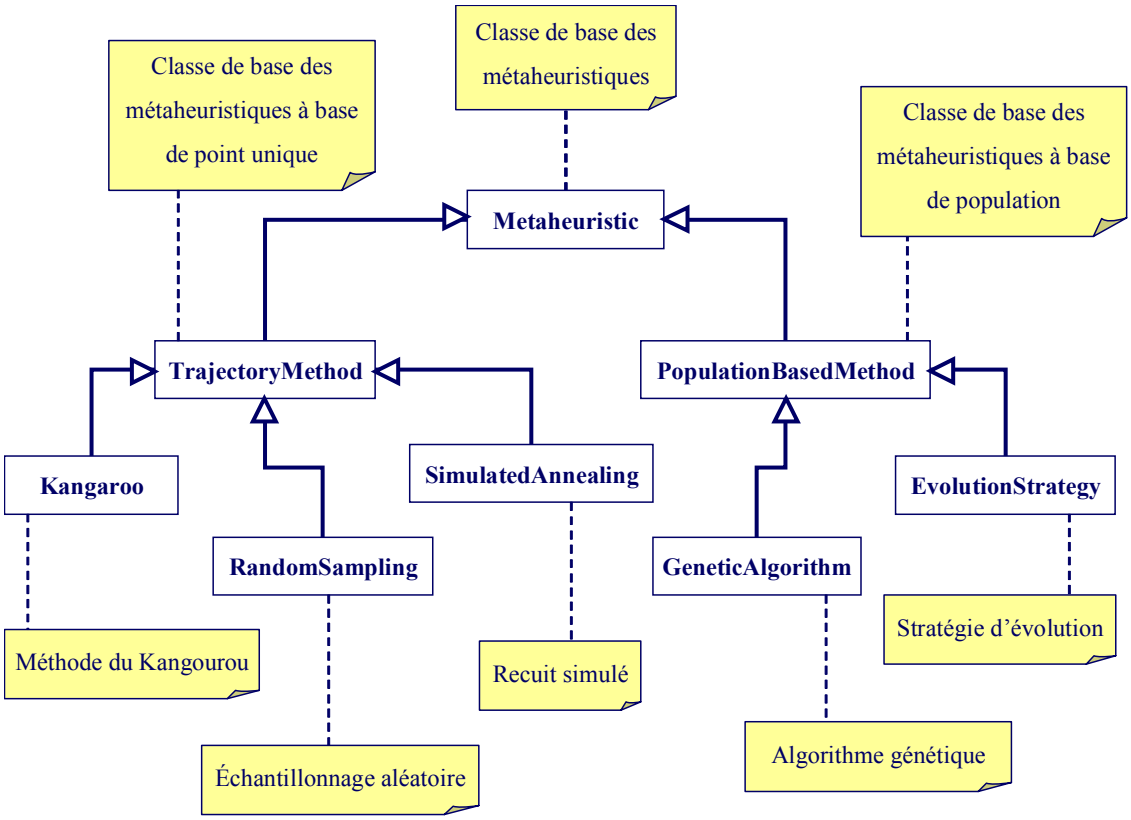


Figure 3.6 - *Modèle d'héritage utilisé pour la conception des métaheuristiques.*

3.2.4.1. Modèle de conception des métaheuristiques

La figure 3.6 montre le modèle d'« héritage » utilisé pour la conception des métaheuristiques. Celui-ci utilise la classification présentée au chapitre 1, en distinguant deux grandes familles : les méthodes à base de point unique (classes héritant de *TrajectoryMethod*) et les méthodes à base de population (classes héritant de *PopulationBasedMethod*). Toutes les métaheuristiques héritent d'une seule et unique classe de base appelée *Metaheuristic*.

Une instance de la classe *Metaheuristic* est composée d'instances des quatre classes suivantes (voir figure 3.7) :

- *SolutionList* : constituée de la liste des meilleures solutions rencontrées au cours de l'optimisation, triées de la plus mauvaise à la meilleure. Cet ordre est maintenu tout au long de l'optimisation et à chaque insertion d'une nouvelle solution dans la liste.
- *Monitor* : permettant de gérer l'arrêt de l'optimisation, les pauses (demandées par l'utilisateur), la mise à jour des vues (courbes et tableaux), ainsi que le mode automatique.

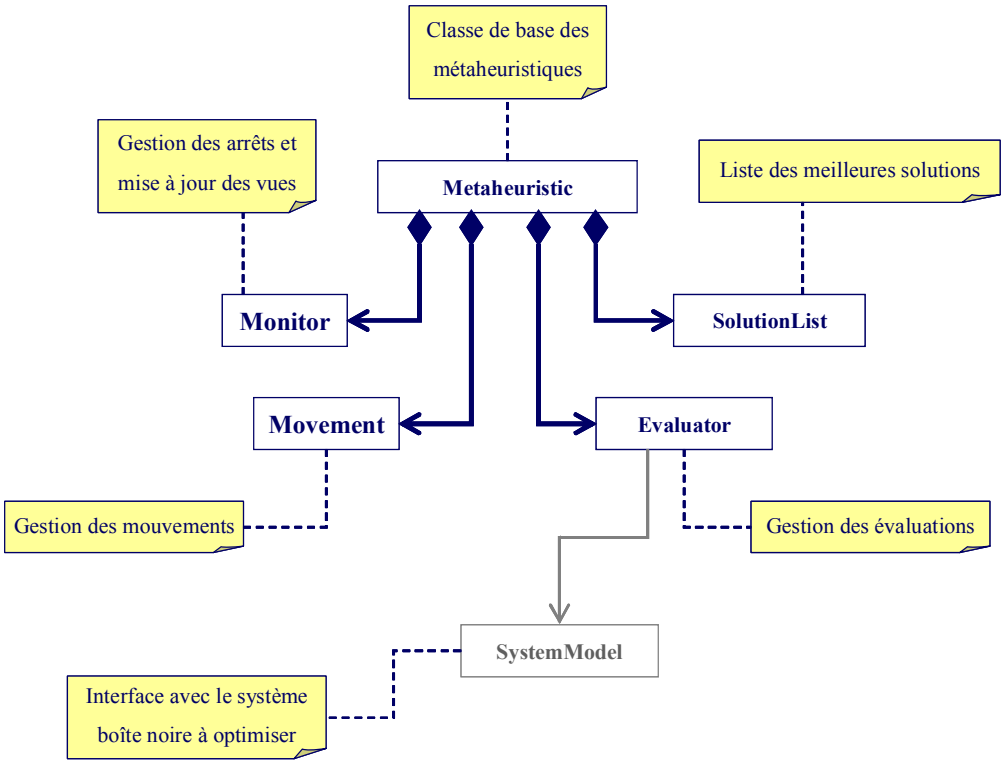


Figure 3.7 - *Composition de la classe Metaheuristic.*

- *Movement* : contenant la configuration des paramètres activés (intervalle de variation et type de variable (réel ou entier)), ainsi que les fonctions permettant d'effectuer un changement, ou de récupérer une information sur les valeurs des paramètres.

- *Evaluator* : contenant la configuration des indicateurs de performance choisis par l'utilisateur (poids associé, sens d'optimisation (minimisation ou maximisation)…), ainsi que les fonctions permettant de calculer la valeur de la fonction objectif et de récupérer les informations sur les valeurs des indicateurs.

En plus des quatre objets ci-dessus, les métaheuristiques peuvent manipuler plusieurs autres types d'objets, qui sont décrits maintenant.

3.2.4.2. Solutions

Les solutions sont des instances de la classe *Solution*. Celle-ci est composée de deux parties (voir figure 3.8) :

- le *modèle de gestion des mouvements* (classe *MovementModel*), permettant de gérer les déplacements dans l'espace des paramètres à régler,
- le *modèle de gestion des évaluations* (classe *EvaluatorModel*), permettant de calculer les valeurs des indicateurs et d'évaluer la valeur de la fonction objectif.

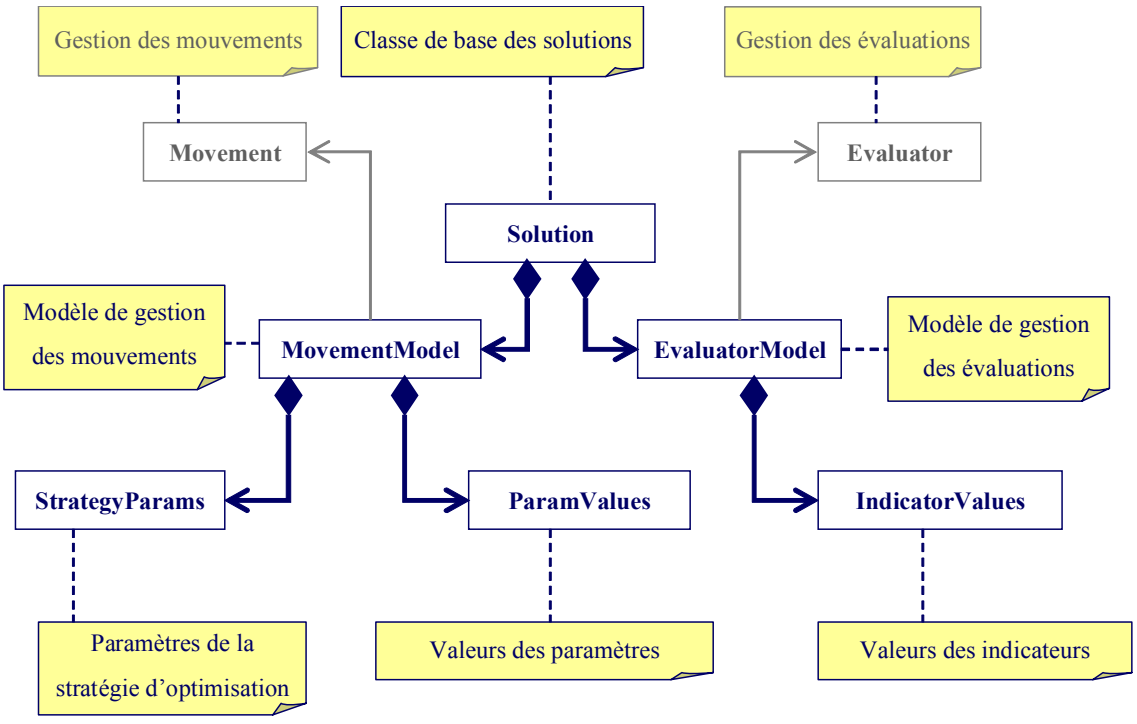


Figure 3.8 - *Modèle de conception des solutions.*

La représentation des populations de solutions est basée sur la classe *SolutionList*, qui contient un ensemble de solutions triées, de la plus mauvaise à la meilleure. Plusieurs fonctions ont été implémentées au sein de cette classe, pour permettre de manipuler des populations :

- éliminer les k plus mauvaises solutions de la population,
- récupérer la meilleure solution,
- créer une nouvelle population, en se basant sur une population existante,
- insérer une nouvelle solution dans la population, en maintenant les solutions triées…

3.2.4.3. Opérateurs

On peut distinguer deux types d'opérateurs :

- *les opérateurs définis sur l'espace des solutions :* deux familles d'opérateurs ont été implémentés, les recombinaisons et les mutations. Un modèle générique pour la conception de ces opérateurs a été implémenté, sur la base du concept d'héritage (voir figure 3.9).
- *les opérateurs définis sur les populations de solutions (voir figure 3.10) :* un seul type d'opérateur de population a été implémenté ici, regroupant les opérateurs de sélection utilisés dans les algorithmes évolutionnaires : roulette de casino, rangement linéaire, sélection uniforme…

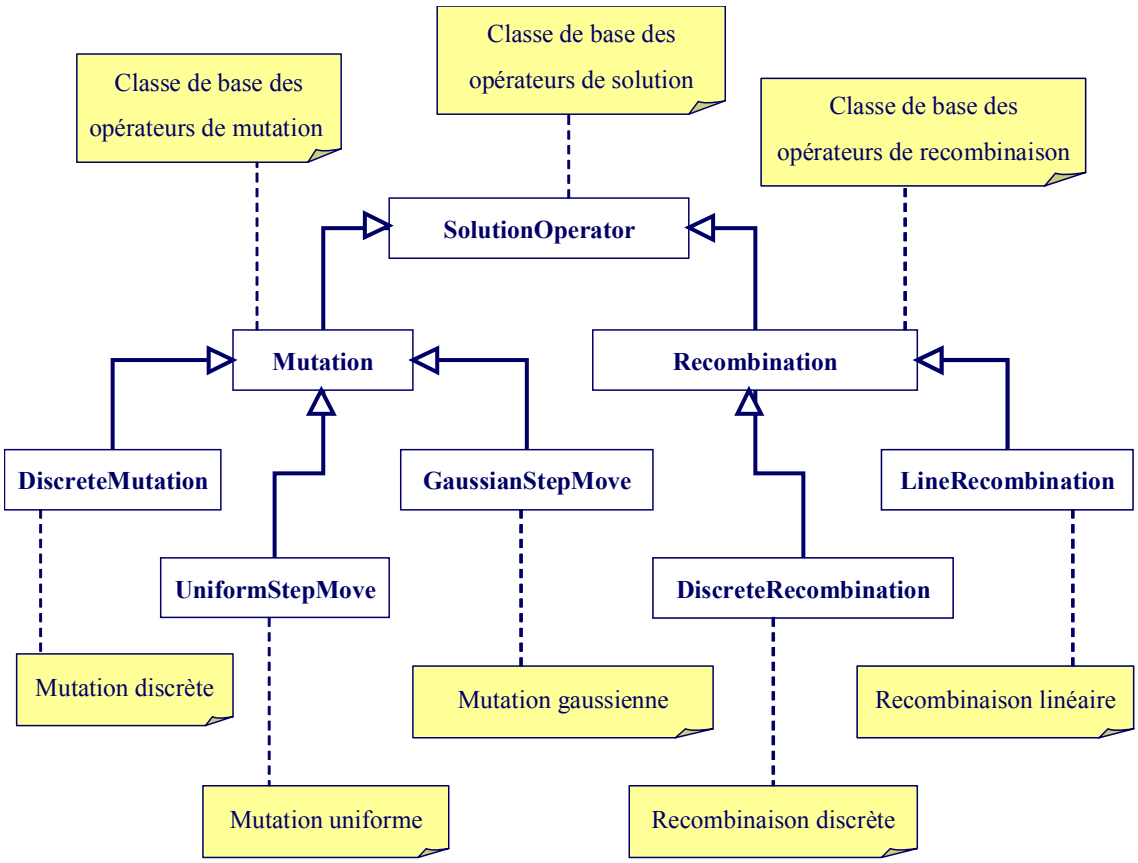


Figure 3.9 - *Modèle d'héritage utilisé pour la conception des opérateurs de mutation et de recombinaison.*

La sélection de paramètres a été incorporée au sein des opérateurs de mutation, selon les deux techniques suivantes :

- la possibilité d'effectuer la mutation, en se restreignant aux paramètres sélectionnés, non sélectionnés ou sans aucune restriction,
- la possibilité de spécifier le nombre de paramètres à muter, à chaque application de l'opérateur de mutation.

3.2.4.4. Heuristiques

Certaines méthodes de recherche aléatoire sont des procédures élémentaires utilisées par plusieurs métaheuristiques, comme l'échantillonnage aléatoire pour l'initialisation de la recherche. Celles-ci héritent toutes d'une même classe de base, appelée *Heuristic*, qui se compose d'objets des classes :

- *SolutionList* : qui contient une liste des meilleures solutions visitées,
- *Criterion* : qui permet de spécifier un critère d'arrêt pour l'heuristique, tel que le nombre de solutions évaluées, le nombre maximum de solutions successives sans amélioration de la fonction objectif…

Parmi les heuristiques implémentées, on trouve :

- la méthode d'échantillonnage aléatoire avec choix de la mutation à utiliser, ainsi que le nombre de paramètres sélectionnés dans les solutions construites,
- la méthode de descente aléatoire, pour améliorer localement une solution donnée, avec choix de la mutation à utiliser et choix du nombre de paramètres à muter à chaque déplacement dans l'espace de recherche,
- les méthodes d'élimination aléatoire en arrière, et de sélection aléatoire en avant (voir les stratégies de sélection de paramètres, chapitre précédent),
- la méthode de recherche par motifs de Hooke et Jeeves (voir la description de cette famille de méthodes au chapitre 1).

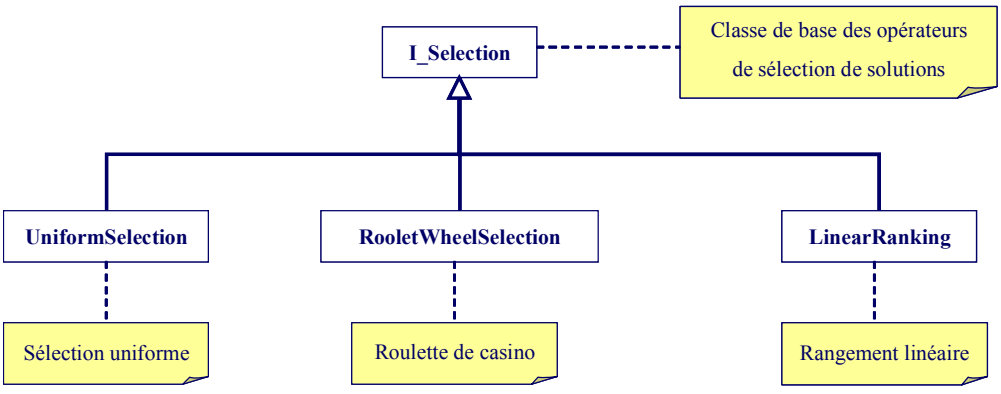


Figure 3.10 - *Modèle d'héritage utilisé pour la conception des opérateurs de sélection.*

3.2.5. Synthèse

Nous avons présenté dans cette partie le prototype *Ortems Optimizer*, qui est un module développé en langage Java, permettant de choisir et de régler les paramètres intervenant en entrée du logiciel d'ordonnancement *Ortems*, sur la base d'une utilisation de métaheuristiques combinées avec des stratégies de sélection. *Ortems Optimizer* n'est pas seulement une application utilisée dans un cadre académique et de recherche, c'est aussi un produit commercialisé, et qui a déjà montré son efficacité dans la résolution pratique de cas industriels (voir annexe C).

Dans ce qui suit, ce prototype sera appliqué à plusieurs exemples de problèmes d'ordonnancement d'atelier, issus de cas industriels réels.

3.3. Logiciel d'ordonnancement *Ortems*

Éditeur de logiciels de planification et ordonnancement, *Ortems* est présent chez plus de 300 clients, installés principalement en France, en particulier dans le monde du « process » (Pharmacie, Chimie et Agro-alimentaire). Fondée à Lyon en 1989, la société *Ortems*, reconnue comme leader européen en planification détaillée (source AMR), fait partie du Groupe FINMATICA.

Ortems propose une suite de solutions pour l'optimisation de la chaîne logistique de production : ordonnancement, programme directeur de production, MRP synchronisé, ainsi qu'un outil d'intégration avec les principaux systèmes d'information du marché : *SAP*, *ORACLE*, ….

La partie suivante, décrit la méthode d'ordonnancement implémentée dans *Ortems*.

3.3.1. Méthode d'ordonnancement

Le moteur d'ordonnancement d'*Ortems* est basé sur une stratégie d'ordonnancement issue d'un travail de recherche original [Al Kazzaz, 1989]. Cette stratégie repose sur l'utilisation d'un graphe orienté, dans lequel les sommets représentent les tâches à ordonnancer et les arcs expriment les contraintes d'enchaînement entre ces tâches.

Le graphe est composé de deux types d'arcs :

- arcs statiques, qui sont connus initialement et sont définis à partir des gammes de fabrication. Ils représentent l'enchaînement temporel entre certaines tâches,
- arcs dynamiques, qui sont construits au cours de l'ordonnancement et modélisent le fait que certaines tâches doivent partager les mêmes ressources (machines, opérateurs, outillages, …etc.) de capacités limitées.

L'ordonnancement est réalisé grâce à un algorithme itératif, qui procède en ordonnançant une seule tâche à chaque itération. L'algorithme s'arrête dès qu'il n'existe plus aucune tâche « ordonnançable » (c'est-à-dire dont tous les prédécesseurs dans le graphe ont été ordonnancés).

Au cours de chaque itération, deux types de choix sont effectués :

- choix d'une tâche à ordonnancer parmi un ensemble de n tâches passant sur le même îlot[19],
- choix de la machine sur laquelle doit être réalisée une tâche donnée, parmi un groupe de machines appartenant au même îlot.

Ces choix sont effectués grâce à des règles de priorité, appelées aussi *règles d'ordonnancement*, si elles concernent un choix de tâche, et *règles de placement*, si elles concernent un choix de machine. Des exemples de règles d'ordonnancement sont : choisir la tâche dont la date de début est la plus proche, le chemin critique le plus petit …. Pour les règles de placement : choisir la machine qui minimise les changements d'outils, la machine disponible au plus tôt…

Étant donné un ensemble C de tâches (resp. de machines), le choix d'une tâche dans C est effectué en combinant un ensemble R_1, R_2, …, R_k de règles élémentaires d'ordonnancement (resp. de placement) grâce à une approche multicritère de type « quasi-ordre lexicographique » [Pomerol et Barba-Romero, 1993]. Les règles sont appliquées l'une après l'autre dans un certain ordre (défini par l'utilisateur), la même règle pouvant être réappliquée plusieurs fois. L'application d'une règle élémentaire R revient à choisir les tâches (resp. machines) *x* qui satisfont :

$h_R(x) \leq h_R^* + m$,

où h_R est une fonction numérique associée à la règle R,

h_R^* est le minimum de la fonction h_R sur C,

et m est un nombre entier positif appelé « marge » (défini par l'utilisateur).

L'idée de combiner plusieurs règles provient du fait que chacune des règles d'ordonnancement et de placement a été conçue pour atteindre certains objectifs, et donne par conséquent de mauvais résultats sur d'autres [Montazeri et Wassenhove, 1990] alors qu'un problème d'ordonnancement d'atelier est en général de nature multicritère [T'Kindt et Billaut, 2002].

L'utilisateur peut intervenir au niveau de l'utilisation des règles, en définissant des *règles composées*, qui correspondent à la spécification des trois éléments suivants :

- sous-ensemble de règles à utiliser,
- ordre dans lequel ces règles doivent être appliquées,
- marge associée à chaque règle.

Une règle composée peut être *locale* ou *globale*. Elle est dite locale si elle n'est applicable qu'à un îlot particulier, et globale si elle s'applique à tous les îlots. Quand l'algorithme se trouve devant un choix de tâche ou de machine pour un certain îlot, il vérifie s'il existe une règle composée locale associée à cet îlot. Si c'est le cas, celle-ci est utilisée, sinon c'est la règle composée globale qui est appliquée.

Avant de pouvoir lancer un ordonnancement, l'utilisateur doit définir une « configuration de règles » qui consiste à spécifier :

---

[19] Un îlot correspond à un groupe de machines au fonctionnement similaire

- soit une règle composée globale,
- soit une règle composée globale et un ensemble de règles composées locales pour certains îlots,
- soit un ensemble de règles composées locales associées à tous les îlots.

A chaque installation du logiciel *Ortems*, une étape importante est la définition d'une bonne configuration de règles, i.e. adaptée à l'atelier de production considéré et aux exigences du client. Ces deux éléments sont évalués grâce à des indicateurs de performance de l'atelier.

3.3.2. Indicateurs de performance

Le processus de production peut être assujetti à une multitude d'objectifs à atteindre, souvent conflictuels. Ces objectifs sont quantifiés par l'intermédiaire d'indicateurs de performance, qui dépendent des caractéristiques de l'atelier et des produits fabriqués. *Ortems* utilise une liste prédéterminée de 23 indicateurs de base (voir annexe A pour une description détaillée des indicateurs disponibles), qui peuvent être :

- liés aux ordres de fabrication (retard, écoulement, séjour …),
- liés aux machines (charge, occupation, temps de réglage …),
- généraux (*Makespan*[20], taux de juste-à-temps …).

Ces indicateurs peuvent être personnalisés par l'utilisateur (par exemple : l'utilisateur peut spécifier l'horizon de temps sur lequel le calcul est effectué, quels ordres de fabrication sont concernés, ainsi que sur quelles machines un indicateur est calculé).

De plus, des indicateurs spécifiques peuvent être construits grâce à une plate-forme de développement livrée avec le logiciel.

3.3.3. Types de contraintes prises en compte par *Ortems*

Le logiciel *Ortems* permet de modéliser une grande variété de contraintes d'ordonnancement :

- *Contraintes de disponibilité de machines.* Ce type de contraintes permet de modéliser le fonctionnement de certaines machines (appelées « cuves »), qui imposent des temps d'indisponibilité (c'est-à-dire que la machine ne peut pas être utilisée durant ces temps) avant et/ou après l'exécution de certaines phases de fabrication. Un exemple est la phase de mélange rencontrée dans certaines industries de « process » (agro-alimentaire, chimie, …). Cette phase consiste à remplir une cuve, mélanger son contenu, puis le vider avant de commencer la tâche suivante. Les deux étapes de remplissage et de vidange sont modélisées

[20] Le « Makespan » d'un ordonnancement est la durée qui sépare le début de la première tâche et la fin de la dernière

par des contraintes de disponibilité de la machine avant et après la phase de mélange.

- *Temps de réglage* (appelés aussi, *temps de changement d'outil ou de paramètre*), qui sont des temps nécessaires entre la fin de la tâche précédente et le début de la tâche suivante sur une même machine, si elles utilisent des outils ou des paramètres différents (les deux notions d'outil et de paramètre peuvent prendre des significations différentes, selon le problème d'ordonnancement considéré). Par exemple, dans le cas où deux tâches de mélange se suivent sur une même cuve : il est nécessaire de faire un nettoyage de la cuve si les deux mélanges sont basés sur des éléments différents ou incompatibles. Les temps de réglage peuvent être variables, en fonction du couple de tâches considérée : dans ce cas, ils sont dits *dynamiques*, sinon les temps sont *fixes*.

- Contraintes de type synchronisation entre deux tâches :

  - synchronisation début – début : les deux tâches doivent débuter en même temps,

  - synchronisation fin – début : la fin de la première tâche coïncide avec le début de la deuxième,

  - lien solide : les deux tâches doivent se succéder sur la même machine et aucune tâche ne peut s'insérer entre elles sur cette machine.

- *Gestion des séries de type campagne*. Une série est une entité permettant de regrouper un ensemble d'ordres de fabrication (OF). Les séries de type « campagne » imposent les règles suivantes sur la gestion des OF qui les composent :

  - tous les OF passent obligatoirement sur les mêmes machines,

  - tous les OF sont exécutés les uns à la suite des autres, sans qu'il soit permis d'insérer entre eux des OF d'autres séries.

  Ces deux règles imposent que la planification du premier OF de la série détermine la planification des OF suivants, ce qui représente une contrainte forte pour l'ordonnancement.

- *Choix entre Gammes alternatives.* A chaque OF sont associées plusieurs gammes différentes permettant de produire des articles fonctionnellement identiques. Le choix entre ces différentes versions est effectué au cours de l'ordonnancement.

- *Temps de transfert entre zones.* Ce temps est celui que met une pièce pour transiter entre deux machines implantées dans des zones géographiques différentes.

- *Chevauchement entre tâches.* Cette contrainte d'ordonnancement est une généralisation des liens de précédence entre deux tâches successives, pour prendre en compte les cas où la seconde tâche peut débuter avant la fin de la première, en respectant une certaine marge de liberté définie par l'utilisateur.

- *Calendriers machine et opérateurs*. Plusieurs calendriers hebdomadaires et annuels peuvent être définis. Ces calendriers permettent de spécifier les horaires de travail des machines et opérateurs.

- *Ressources limitées.* L'utilisation de ressources à capacité limitée permet de prendre en compte un ensemble de ressources complémentaires, différentes des machines, qui sont nécessaires pour la réalisation des tâches. Ces ressources, qui sont réutilisables, existent en nombre limité et sont partagées par l'ensemble des tâches qui les nécessitent. Elles sont donc à prendre en compte dans l'ordonnancement à capacité finie. Ainsi, si la demande en ressources est supérieure au nombre de ressources disponibles, les tâches sont décalées automatiquement dans le temps, jusqu'à la prochaine disponibilité de ressources.
- *Contraintes sur Machines.* Trois types de contraintes sur les machines sont gérés par *Ortems* :

  - *Machines incompatibles.* Deux machines sont incompatibles si elles ne peuvent fonctionner en même temps, c'est-à-dire que la planification d'une tâche sur une des deux machines entraîne l'impossibilité de placer une autre tâche sur la deuxième machine dans la même période de temps.

  - *Machine de type Run.* Une machine qui travaille en mode *Run* traite une seule tâche à la fois en la découpant en lots. La taille de ces lots est fonction de la capacité de chargement de la machine. Sur ces machines, les tâches sont caractérisées par une capacité de chargement qui définit la taille du lot et une durée de réalisation d'un lot.

  - *Machines de type batch.* Une machine qui travail en mode *batch* travaille par cycle et peut traiter plusieurs opérations dans le même cycle. Dans ce mode, on peut traiter plusieurs OF en même temps. Une machine batch est caractérisée par une capacité minimale et maximale de chargement.

3.4. Expérimentations numériques

Les points suivants seront analysés dans cette partie :

- *comparaison avec l'échantillonnage aléatoire :* bien qu'elle soit la plus simple parmi les méthodes non déterministes, la méthode d'échantillonnage aléatoire peut donner, sur certains problèmes, des résultats aussi bons voire meilleurs que d'autres algorithmes plus complexes [Galindo-Legaria, 1994]. Il est donc important de justifier l'intérêt d'utiliser les algorithmes qui seront expérimentés dans ce chapitre.
- *analyse expérimentale de la sélection de paramètres :* une étude comparative sera effectuée entre les méthodes avec et sans sélection de paramètres, pour analyser les avantages et inconvénients liés à l'introduction de la sélection de paramètres dans les métaheuristiques expérimentées.
- *Comparaison avec le paramétrage manuel* (basé sur l'expérience),
- *étude comparative entre les approches proposées*.

Bien qu'ils soient intéressants, les points suivants ne seront pas traités ici :

- le comportement asymptotique (en raison du coût trop élevé en temps de calcul d'une telle expérimentation), notre intérêt portera sur le comportement en un temps raisonnable,
- l'optimisation multicritère (qui nécessite un travail important avec les décideurs concernés par chaque cas étudié).

Néanmoins, nous pensons que le deuxième point (aspect multicritère) est essentiel et mérite d'être analysé pour vérifier la validité de nos approches. Il fait partie de nos perspectives de recherche, que nous exposerons dans la partie conclusion et perspectives.

3.4.1. Description des problèmes tests

Les expérimentations numériques ont été réalisées sur plusieurs problèmes tests, qui seront décrits ici. L'ordonnancement utilise 57 paramètres, qui sont les poids associés aux règles de priorité, prenant leurs valeurs dans l'intervalle [0, 1]. L'utilisation des stratégies de sélection doit permettre de choisir un nombre réduit de paramètres, afin de limiter le temps d'ordonnancement (qui croît avec le nombre de paramètres non nuls). Une liste des règles de priorité utilisées par le logiciel *Ortems* est présentée en annexe B. Les 57 règles de priorité standard d'*Ortems* sont constituées de 18 règles de placement et 39 règles d'ordonnancement. Parmi les règles d'ordonnancement, 5 sont spécifiques aux problèmes avec des gammes alternatives, comme dans le cas Chimie (voir la présentation de ce cas dans la partie 3.1.2).

Le nombre de paramètres considérés est :

- 52 pour les cas Agro1, Meca et Agro2, associés aux 18 règles de placement et 34 règles d'ordonnancement autres que celles spécifiques aux gammes alternatives,
- 55 pour le cas Chimie, correspondant aux 18 règles de placement et 37 règles d'ordonnancement (dont 5 sont celles spécifiques aux gammes alternatives).

3.4.1.1. Cas Agro1

Le premier problème test est un atelier spécialisé dans le mélange et le conditionnement pour l'industrie agroalimentaire. Cet atelier est composé d'un parc de 14 machines utilisées dans les différentes étapes du processus de production.

Les principales contraintes d'ordonnancement sont (voir la description du logiciel Ortems dans la partie 2, pour une description des principaux types de contraintes prises en compte par *Ortems*) :

- l'existence de contraintes de disponibilité de machines,
- l'existence de contraintes de type synchronisation,
- prise en compte des temps de réglage dynamiques,
- calendriers machines.

L'instance considérée concerne l'ordonnancement de 70 ordres de fabrication, tous basés sur une gamme de trois opérations. Les principales statistiques descriptives pour la durée d'une opération dans l'atelier sont données dans le tableau 3.1.

L'objectif de l'ordonnancement est de réduire la durée totale de l'ordonnancement (ou *makespan*).

	DT
Moyenne	2.09 h
Médiane	2 h
Minimum	0 h
Maximum	8 h
1er quartile	1 h
2ème quartile	2.29 h
Étendue	8 h
Écart type	1.34 h

Tableau 3.1 (Cas Agro1) - *Statistiques descriptives de la durée d'une tâche, pour le cas Agro1 (DT : Durée d'une tâche en heures).*

3.4.1.2. Cas Chimie

Le second problème test concerne une compagnie de production de parfums et arômes. Les données de l'environnement de production sont mises à jour au début de chaque semaine, et un nouvel ordonnancement est réalisé avec le logiciel *Ortems*. L'atelier de production est composé d'un parc de 135 machines et 289 ordres de fabrication sont considérés ici.

Les principales contraintes d'ordonnancement sont les suivantes :

- synchronisation entre tâches,
- contraintes de disponibilité de machines,
- gestion des séries de type campagne,
- choix entre gammes alternatives,
- calendriers machines.

Après discussions avec les décideurs de l'entreprise, les indicateurs suivants ont été choisis (les nombres entre parenthèses représentent les poids, associés aux indicateurs, apparaissant dans l'écriture de la fonction objectif agrégée) :

- retard moyen (100%),
- makespan (10%).

3.4.1.3. Cas Méca

On considère ici une usine spécialisée dans la fabrication de produits d'étanchéité pour l'industrie mécanique. Son objectif est d'optimiser le taux de service et le « makespan », avec des poids respectifs de 100 % et 10 %. L'instance considérée est

constituée d'un parc de 81 machines, sur lesquelles doivent être ordonnancées 5857 tâches (appartenant à 1302 ordres de fabrication).

Les principales contraintes sont les suivantes :

- temps de transfert entre zones,
- chevauchement entre tâches,
- calendriers machines.

	NT/OF	DT
Moyenne	3.60	3.77 j
Médiane	3	2.33 j
Minimum	2	0 j
Maximum	9	32.57 j
1er quartile	2	0 j
2ème quartile	4	5.17 j
Étendue	7	32.57 j
Écart type	1.56 j	4.8 j

Tableau 3.2 (Cas Chimie) - *Statistiques descriptives du nombre d'opérations par ordre de fabrication et de la durée d'une opération, pour le cas Chimie (NT/OF : Nombre de tâches par ordre de fabrication, DT : Durée d'une tâche en jours).*

3.4.1.4. Cas Agro2

Ce dernier problème test consiste à réaliser l'ordonnancement de la production d'une entreprise appartenant au secteur agroalimentaire. L'instance considérée est constituée d'un parc de 42 machines, 652 tâches à ordonnancer, avec les objectifs suivants (les chiffres entre parenthèses sont les poids associés dans l'écriture de la fonction objectif) : taux de service (100 %), « makespan » (10 %) et temps de réglage (1 %).

Les principales contraintes d'ordonnancement sont les suivantes :

- temps de réglage,
- calendriers machines.

3.4.2. Plan d'expériences

3.4.2.1. Introduction

Étant donné le coût élevé et la complexité de mise en œuvre de ce type d'expériences, les expérimentations ont été divisées en deux étapes, avec des objectifs différents. Dans une première série de tests, l'objectif est de faire une analyse statistique des approches proposées sur deux petits problèmes, en limitant le temps d'exécution à 30 minutes. La deuxième partie des expérimentations permettra de compléter et de valider les résultats obtenus avec la première série de tests, en analysant le

comportement des approches étudiées lorsque le test d'arrêt n'est plus le temps d'exécution, mais le nombre d'évaluations de la fonction objectif (ou nombre d'ordonnancements), et aussi en réalisant des expériences sur d'autres cas industriels. Dans cette deuxième partie, une comparaison avec le *paramétrage manuel* (paramétrage obtenu par un expert) est effectuée.

3.4.2.2. Choix des méthodes expérimentées et réglage de leurs paramètres

Parmi les approches présentées au chapitre 2, nous avons choisi de tester les trois méthodes suivantes :

- la méthode du kangourou (KANG), qui est l'une des plus simples parmi les méthodes à base de solution unique,
- deux versions simples des algorithmes à base de population de points : un algorithme génétique à codage réel (AG) et une stratégie d'évolution (SE).

D'autres métaheuristiques entrent dans le cadre général des approches présentées au chapitre 2, comme la méthode tabou ou le recuit simulé, mais la plupart de ces méthodes sont plus complexes, ou nécessitent un effort important pour incorporer les différentes stratégies de sélection, et pour régler les différents paramètres utilisés par la méthode. Notons aussi que le problème de « choix d'une métaheuristique » reste un sujet ouvert [Dréo et al., 2003], les utilisateurs font souvent appel à leur savoir-faire et à leur expérience, plutôt qu'à l'application fidèle de règles bien établies.

Dans le but d'analyser l'effet des stratégies de sélection de paramètres, nous utiliserons deux versions de chaque méthode d'optimisation retenue : une version avec des stratégies de sélection de paramètres, et une autre sans sélection. Nous utiliserons les notations suivantes :

- KANG, AG, SE : méthodes sans sélection,
- KANGS, AGS, SES : méthodes avec sélection.

Les tableaux 3.3, 3.4 et 3.5 donnent les valeurs des paramètres intervenant dans les trois métaheuristiques choisies. Ces valeurs ont été ajustées de la manière suivante :

- les paramètre λ (nombre d'enfants) et μ (nombre de parents) utilisés par SE et SES ont été fixés aux valeurs usuelles λ=50 et μ=7.
- Les paramètres des mutations : écart type (σ) et probabilité de sélection de paramètres (ρ), ont été déterminés en utilisant une série d'expériences numériques sur le cas Agro1 pour les valeurs suivantes : $\sigma \in \{0.1, 0.2\}$ et $\rho \in \{0, 0.1, 0.2, 0.3\}$.
- Les autres paramètres ont été fixés de manière empirique.

3.4.2.3. Première série de tests

Les expérimentations numériques ont été effectuées sur un ordinateur personnel doté d'un processeur Pentium 4 fonctionnant avec une vitesse d'horloge de 2.4 GHz, avec une mémoire vive de 1 Go, et sous le système d'exploitation Windows 2000.

Étant donné le caractère non déterministe des algorithmes, il est nécessaire de répéter les expériences pour avoir des données statistiquement interprétables, ce qui est très coûteux en temps de calcul. Nous avons donc choisi de mener une première série de tests sur les deux problèmes tests Agro1 et Chimie, qui demandent des temps d'ordonnancement relativement faibles (de l'ordre de la seconde). Chaque test est répété 30 fois et consiste à lancer un algorithme d'optimisation pour 30 minutes.

Nombre de paramètres sélectionnés dans la population initiale (SI)	8
Nombre de sauts	3
Nombre maximum d'arrêts avant un saut	60
Nombre maximum de paramètres sélectionnés (SM)	15
Probabilité d'ajout de paramètre (SM)	0.25
Taille de l'échantillon initial	40
Taille maximum du pas	20

Tableau 3.3 - *Valeurs des paramètres pour les algorithmes du Kangourou KANG et KANGS.*

Taille de la population	50
Nombre de recombinaisons	50
Écart type de la mutation gaussienne	0.1
Nombre minimum de paramètres sélectionnés (SR)	4
Nombre maximum de paramètres sélectionnés (SR) (SM)	15
Nombre de paramètres sélectionnés dans la population initiale (SI)	8
Probabilité d'ajout de paramètres (SM)	0.2
Nombre maximum de recombinaisons par couple de solutions (SR)	5

Tableau 3.4 - *Valeurs des paramètres pour les algorithmes génétiques AG et AGS.*

Nombre d'enfants (λ)	50
Nombre de parents (μ)	7
Ecart type de la mutation gaussienne	0.1
Nombre minimum de paramètres sélectionnés (SR)	4
Nombre maximum de paramètres sélectionnés (SR)	15
Nombre de paramètres sélectionnés dans la population initiale (SI)	8
Probabilité d'ajout de paramètres (SM)	0.2
Nombre maximum de recombinaisons par couple de solutions (SR)	5
Stratégie de remplacement	$\lambda + \mu$

Tableau 3.5 - *Valeurs des paramètres pour les stratégies d'évolution SE et SES.*

Deux types d'outils statistiques sont utilisés pour comparer les performances des différentes méthodes :

- les représentations graphiques des statistiques élémentaires : pour représenter les caractéristiques de tendance centrale et de dispersion, nous utiliserons des

graphiques de type *boîte à moustaches*, appelés aussi *Box Plot*[21] (voir figure 3.11).

- le test U de Mann-Whitney [Sheskin, 1997] : ce test est l'un des plus puissants pour comparer les moyennes de deux échantillons indépendants. Notons qu'une variante de ce test a été aussi conseillée comme outil de choix pour comparer des méthodes itératives non déterministes [Taillard, 2001]. L'hypothèse nulle consiste à supposer que les deux échantillons ont la même valeur moyenne. Rejeter cette hypothèse revient à accepter qu'il existe une différence significative entre les moyennes des deux échantillons comparés. Pour rejeter ou non l'hypothèse nulle, on utilisera la notion de *seuil de signification* d'un test (appelé aussi *niveau p* ou *erreur de première espèce*), qui représente la probabilité d'erreur associée au rejet de l'hypothèse nulle (c'est-à-dire accepter l'hypothèse qu'il existe une différence significative entre les moyennes des deux échantillons, alors que cette hypothèse est fausse).

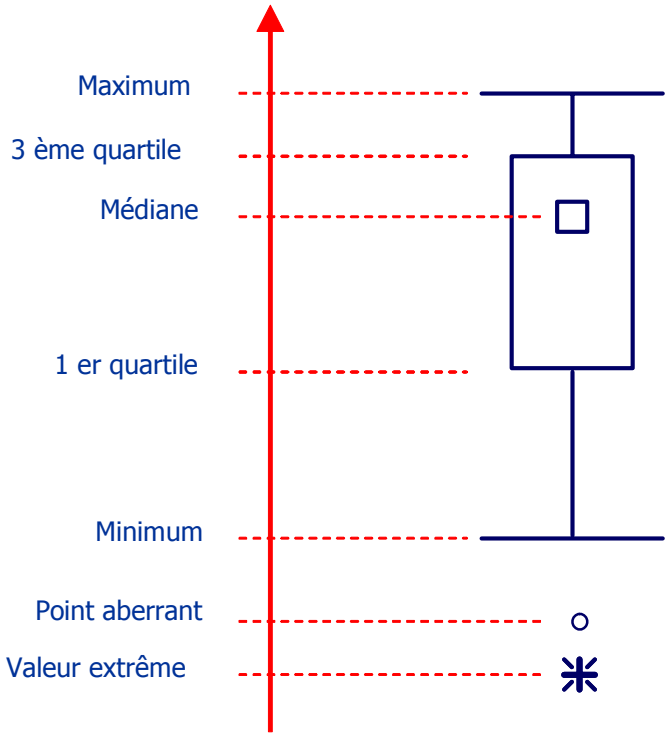


Figure 3.11 - *Représentation graphique de type boîte à moustaches.*

Le test U est inclus dans la plupart des logiciels statistiques modernes (SAS, SPSS, STATISTICA, …). C'est aussi un test non paramétrique, c'est-à-dire ne faisant aucune hypothèse sur les distributions des échantillons à comparer. Cette dernière propriété est importante dans notre cas : les expériences étant très coûteuses en temps de calcul, elles ne peuvent être répétées suffisamment pour pouvoir supposer des distributions gaussiennes.

[21] Ce terme a été utilisé pour la première fois par Tukey en 1970.

3.4.2.4. Seconde série de tests

Afin de valider et de compléter les résultats obtenus sur les deux premiers problèmes tests, une seconde série d'expériences a été menée, portant sur les quatre problèmes tests décrits précédemment. Contrairement aux premières expériences, celles-ci n'ont pas été répétées, car elles nécessitent des temps de calcul trop élevés. Chaque expérience consiste à exécuter une méthode d'optimisation, et à l'arrêter au bout de 700 évaluations de la fonction objectif.

Cette seconde série de tests a été réalisée sur un ordinateur portable Pentium 4, fonctionnant avec 3 GHz de vitesse d'horloge, 1 Go de mémoire vive et utilisant le système d'exploitation Windows XP (version 2002).

3.4.3. Analyse des résultats

Les figures 3.12 et 3.13 présentent l'évolution au cours du temps (30 minutes d'exécution) de la fonction objectif pour les deux versions (avec et sans sélection) des méthodes : Kangourou, stratégie d'évolution et algorithme génétique.

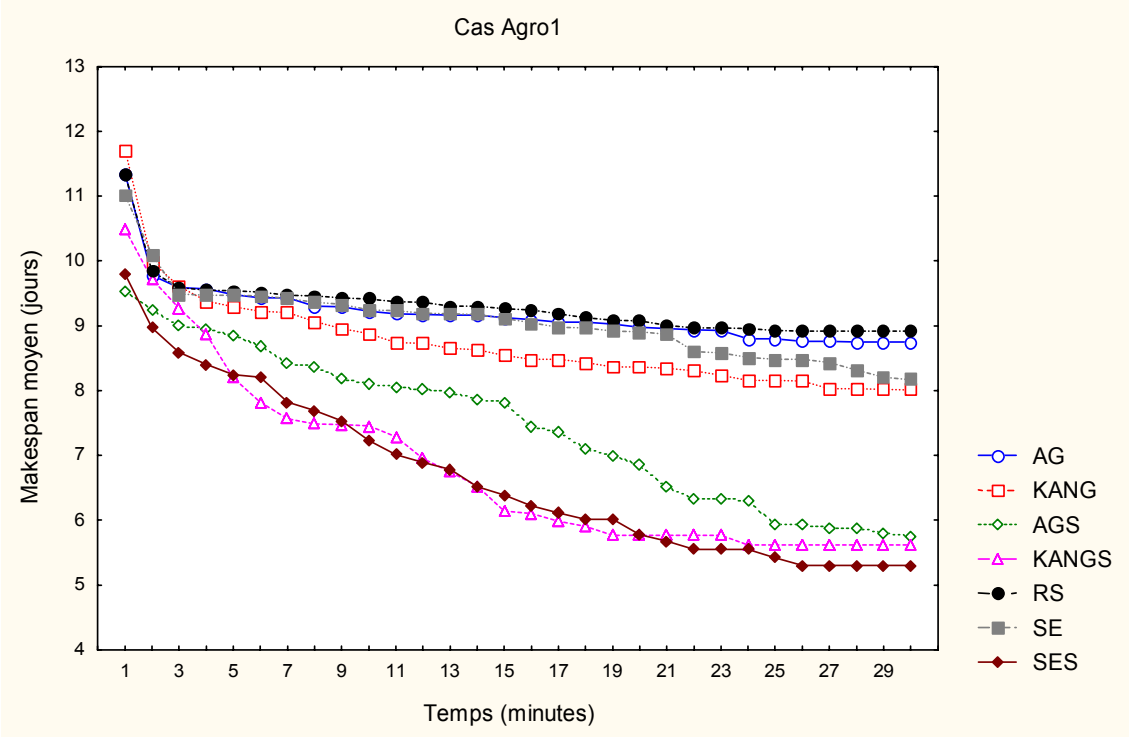


Figure 3.12 (cas Agro1) - *Évolution de la valeur moyenne de la fonction objectif, pour 30 minutes d'exécution.*

En plus de ces méthodes, on retrouve aussi la méthode d'échantillonnage aléatoire (« Random Sampling » - RS), qui permettra de vérifier la validité du théorème « no free lunch » pour les deux problèmes analysés, et de justifier l'utilisation de métaheuristiques. En effet, il est inutile d'utiliser une méthode sophistiquée, si elle ne permet pas d'obtenir des résultats meilleurs qu'un échantillonnage aléatoire.

Les tableaux 3.6 et 3.7 présentent les seuils de signification du test de Mann-Whitney pour la comparaison des moyennes des valeurs de la fonction objectif, pour chaque couple de méthodes : (RS, AG), (RS, AGS), (RS, SE)…

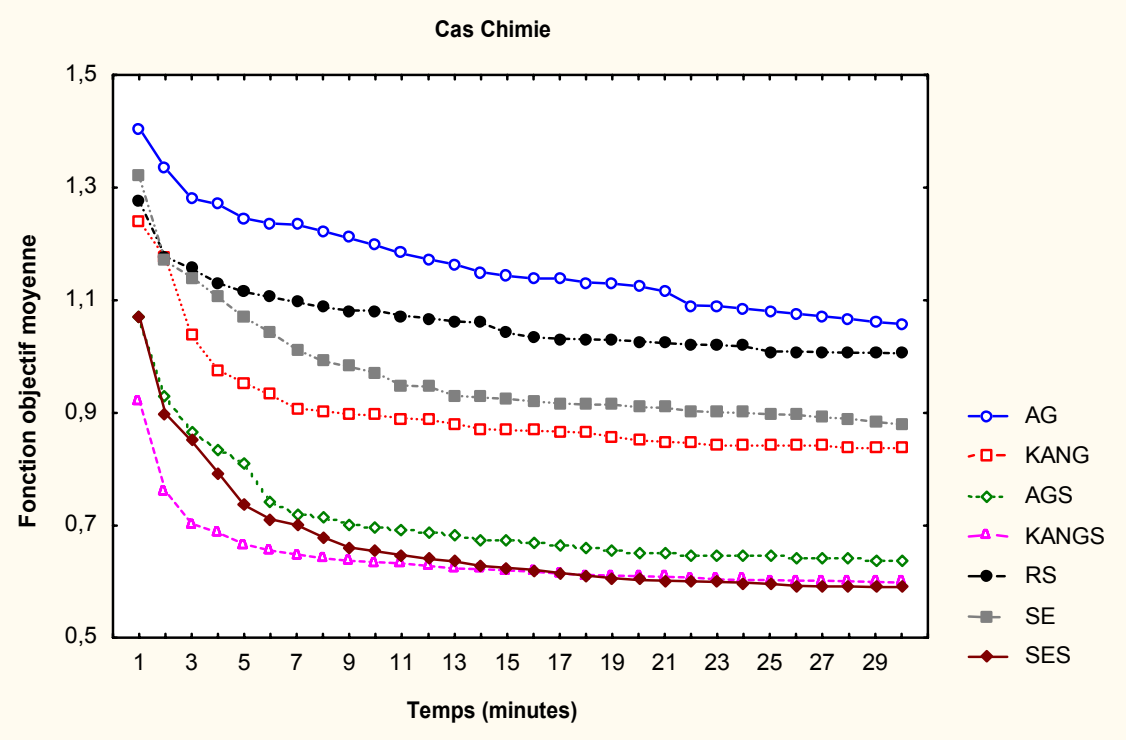


Figure 3.13 (cas Chimie) - *Évolution de la valeur moyenne de la fonction objectif, pour 30 minutes d'exécution.*

3.4.3.1. Comparaison avec l'échantillonnage aléatoire

A partir des figures 3.12 et 3.13, on constate que la version sans sélection de l'algorithme génétique est aussi mauvaise qu'un échantillonnage aléatoire pour le cas Agro1, et plus mauvaise pour le cas Chimie. Ces constatations sont confirmées aussi par le test U de Mann-Whitney (voir tableaux 3.6 et 3.7) :

- Cas Agro1 : le seuil de signification est de 0.620, ce qui implique qu'on ne peut rejeter l'hypothèse nulle selon laquelle les valeurs moyennes des fonction objectifs sont identiques. AG et RS ont donc les mêmes performances moyennes.
- Cas Chimie : le seuil de signification est de 0.000, l'hypothèse nulle est donc rejetée. AG est donc plus mauvais que RS en moyenne.

L'analyse des tableaux 3.6 et 3.7 montre aussi que toutes les autres méthodes utilisées KANG, KANGS, SE, SES et AGS sont significativement meilleures que l'échantillonnage aléatoire (seuil de signification du test de Mann-Whitney inférieur à 0.001).

En analysant la seconde série de tests, il apparaît que pour certains cas (Agro1 et Meca), les algorithmes sans sélection AG, KANG et SE sont au mieux aussi mauvais

qu'un échantillonnage aléatoire. D'un autre côté, les résultats obtenus en utilisant les méthodes avec sélection de paramètres sont nettement meilleurs qu'un échantillonnage aléatoire.

	RS	**AG**	**AGS**	**SE**	**SES**	**KANG**	**KANGS**
RS		0.620	0.000	0.000	0.000	0.000	0.000
AG			0.000	0.030	0.000	0.012	0.000
AGS				0.000	0.092	0.000	0.237
SE					0.000	0.077	0.000
SES						0.000	0.723
KANG							0.000

Tableau 3.6 (cas Agro1) - *Seuil de signification du test U de Mann-Whitney appliqué à la valeur de la fonction objectif, pour 30 minutes d'exécution des différentes méthodes d'optimisation.*

	RS	AG	AGS	SE	SES	KANG	KANGS
RS		0.000	0.000	0.000	0.000	0.000	0.000
AG			0.000	0.000	0.000	0.000	0.000
AGS				0.000	0.000	0.000	0.000
SE					0.000	0.000	0.000
SES						0.000	0.280
KANG							0.000

Tableau 3.7 (cas Chimie) - *Seuil de signification du test U de Mann-Whitney appliqué à la valeur de la fonction objectif, pour 30 minutes d'exécution des différentes méthodes d'optimisation.*

3.4.3.2. Analyse de l'effet des stratégies de sélection de paramètres

Afin d'étudier l'effet des stratégies de sélection de paramètres sur les performances des méthodes proposées, nous avons fait, pour chacune d'entre elles, une étude comparative entre les deux versions (avec et sans sélection), en analysant les fonctions objectifs et les nombres de paramètres sélectionnés. De ces expériences, il ressort les points suivants :

- la valeur de la fonction objectif est nettement meilleure dans le cas d'une sélection de paramètres. D'après les figures 3.14 et 3.15, les méthodes AGS, SES et KANGS sont plus performantes, par rapport à leurs versions sans sélection : meilleure moyenne, étendues des intervalles interquartiles plus faibles. Cette constatation est aussi validée par le test U de Mann Whitney, qui montre qu'il existe une différence significative (seuil de signification inférieur à 0.001) entre les versions avec et sans sélection.

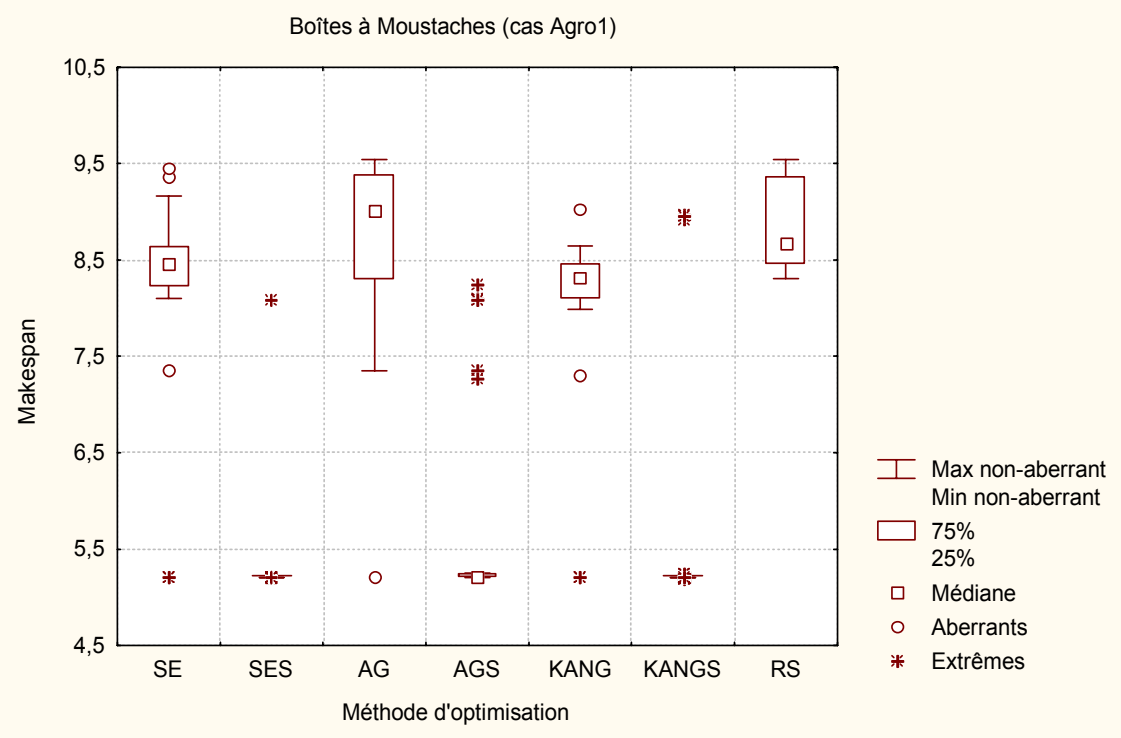


Figure 3.14 (cas Agro1) - *Valeur de la fonction objectif obtenue après 30 minutes d'exécution des différentes méthodes d'optimisation.*

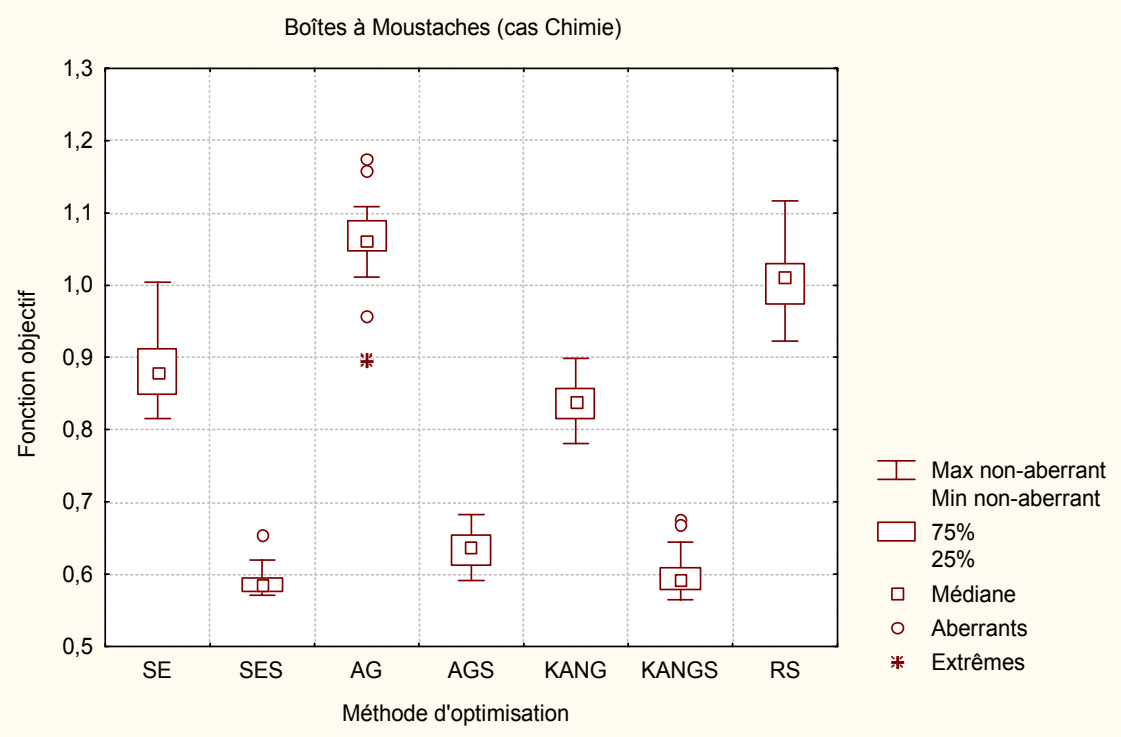


Figure 3.15 (cas Chimie) - *Valeur de la fonction objectif obtenue après 30 minutes d'exécution des différentes méthodes d'optimisation.*

- Le constat précédent s'explique aussi par un nombre plus important d'évaluations de la fonction objectif dans les méthodes avec sélection (voir les figures 3.16 et 3.17), et donc une exploration plus importante de l'espace de recherche.

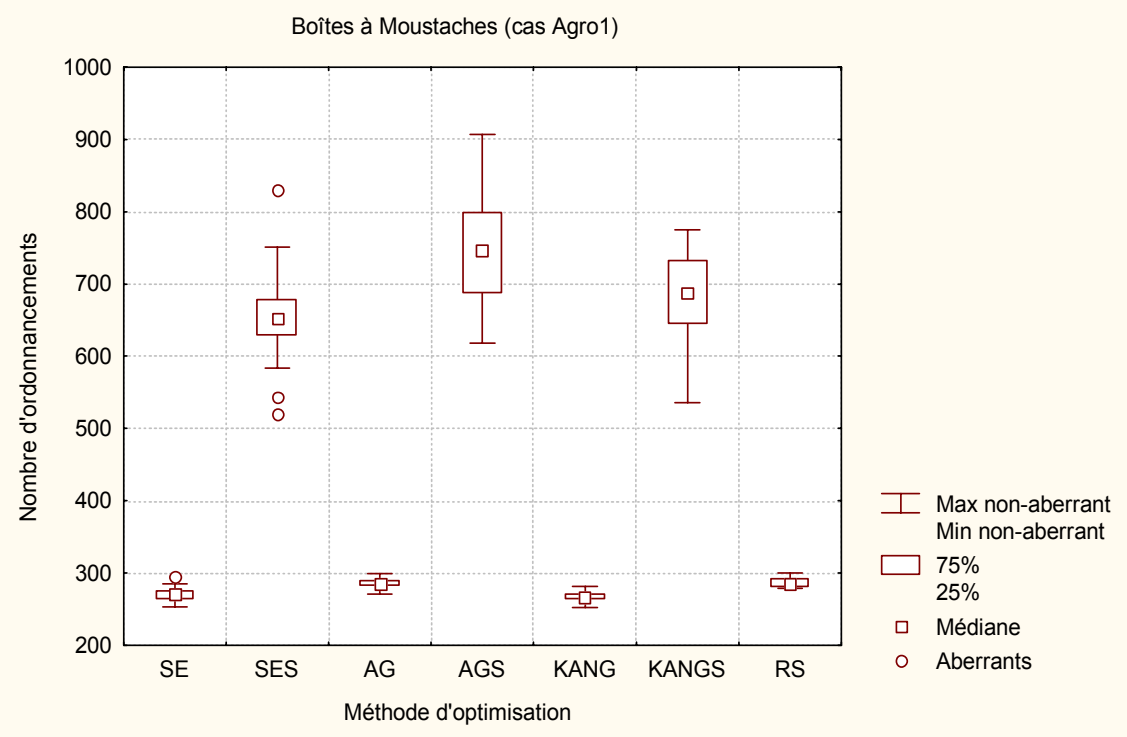


Figure 3.16 (cas Agro1) - *Boîtes à moustaches du nombre d'ordonnancements, pour 30 minutes d'exécution des différentes méthodes d'optimisation.*

- En plus du fait qu'elles sont plus performantes, les méthodes avec sélection offrent l'avantage d'utiliser un nombre beaucoup plus faible de paramètres, en éliminant les paramètres inutiles et non pertinents au cours de l'optimisation (voir les figures 3.18 et 3.19). La durée de l'ordonnancement est donc plus faible, car celle-ci varie de manière croissante en fonction du nombre de paramètres sélectionnés.
- Les trois points précédents sont aussi validés par la deuxième série de tests, qui montre que, même si on donne plus de temps de calcul aux algorithmes d'optimisation sans sélection, leurs performances restent mauvaises (voir figures 3.20, 3.21, 3.22 et 3.23).

Par conséquent, les méthodes avec sélection apparaissent comme nettement meilleures que celles sans sélection, pour les problèmes tests étudiés.

3.4.3.3. Comparaison avec le paramétrage manuel

En comparant les résultats obtenus dans la deuxième série de tests avec ceux du réglage manuel obtenu par un expert (voir les tableaux 3.8, 3.9, 3.10 et 3.11), on constate :

- Cas Agro1 : le makespan passe de 10 jours à 5.23 jours (environ 50 % d'amélioration) avec les méthodes KANGS et SES.
- Cas Chimie : le retard moyen des ordres de fabrication passe de 79 jours à 20.29 jours (une réduction d'environ 70 % du retard moyen) et le makespan passe de 413 jours à 286 jours.
- Cas Meca : le taux de service est amélioré d'environ 47 % (de 31 % à 45.5 %) avec KANGS et d'environ 50 % (de 31 % à 46.66 %) avec SES.
- Cas Agro2 : le taux de service passe de 85 % à 92.79 % et le temps de réglage est réduit de 4 jours (il passe de 37 jours avec le paramétrage manuel à 33 jours avec KANGS).

Ainsi, pour les quatre problèmes étudiés, l'indicateur de performance le plus important est amélioré, avec des taux d'amélioration allant de 10 à 70 %. Dans une application à un cas industriel réel (voir annexe C), l'amélioration constatée sur le taux de service a été de 50 %. Cet écart important entre les résultats obtenus par un expert, et ceux déterminés par un paramétrage automatique, a été aussi constaté sur plusieurs autres cas d'applications [Talbi et al., 2002, 2003, 2004a et 2004b]. Ceci montre la difficulté du paramétrage manuel, même pour des gens expérimentés, pour les raisons suivantes :

- difficulté de généralisation de l'expérience acquise,
- nombre important de règles disponibles,
- complexité des problèmes d'ordonnancement rencontrés en pratique.

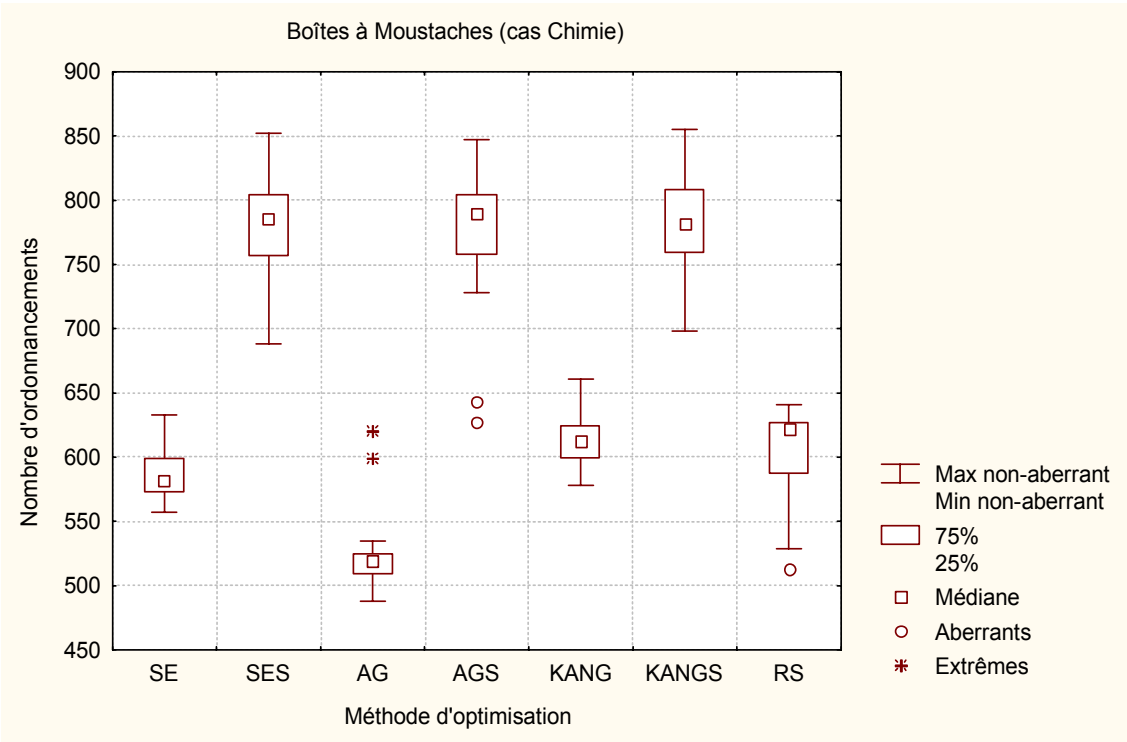


Figure 3.17 (cas Chimie) - *Boîtes à moustaches du nombre d'ordonnancements, pour 30 minutes d'exécution des différentes méthodes d'optimisation.*

Figure 3.18 (cas Agro1) - *Nombre de paramètres sélectionnés dans la solution obtenue, après 30 minutes d'exécution des différentes méthodes d'optimisation.*

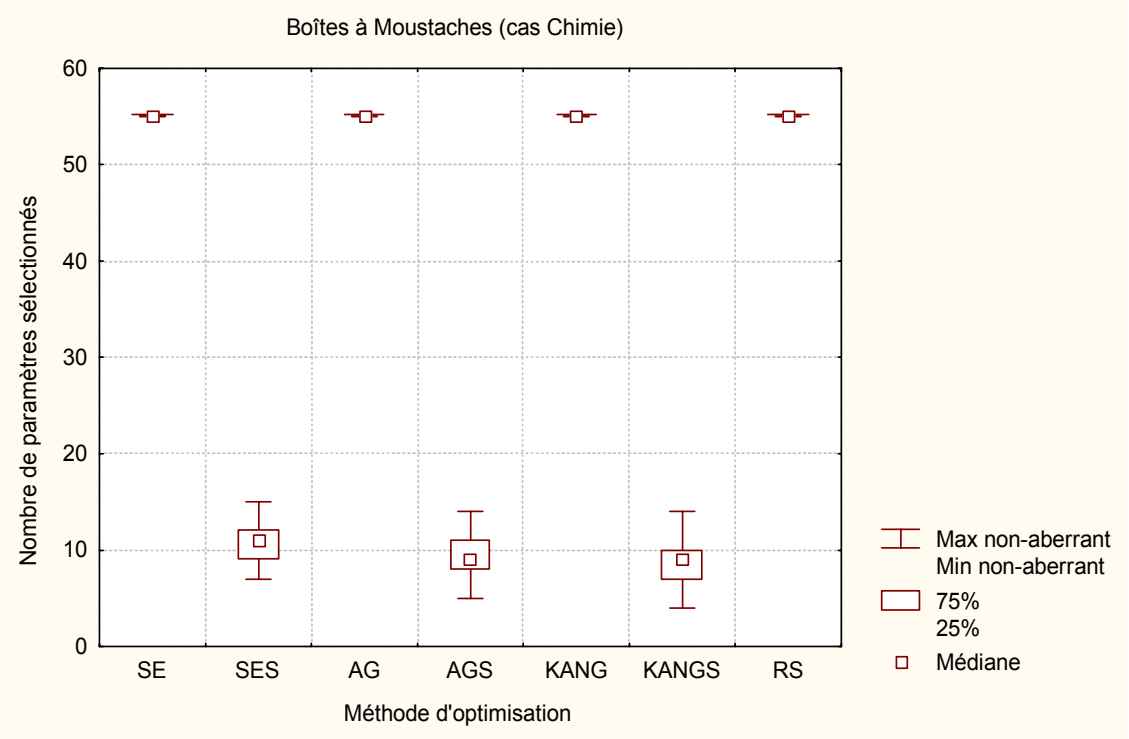


Figure 3.19 (cas Chimie) - *Nombre de paramètres sélectionnés dans la solution obtenue, après 30 minutes d'exécution des différentes méthodes d'optimisation.*

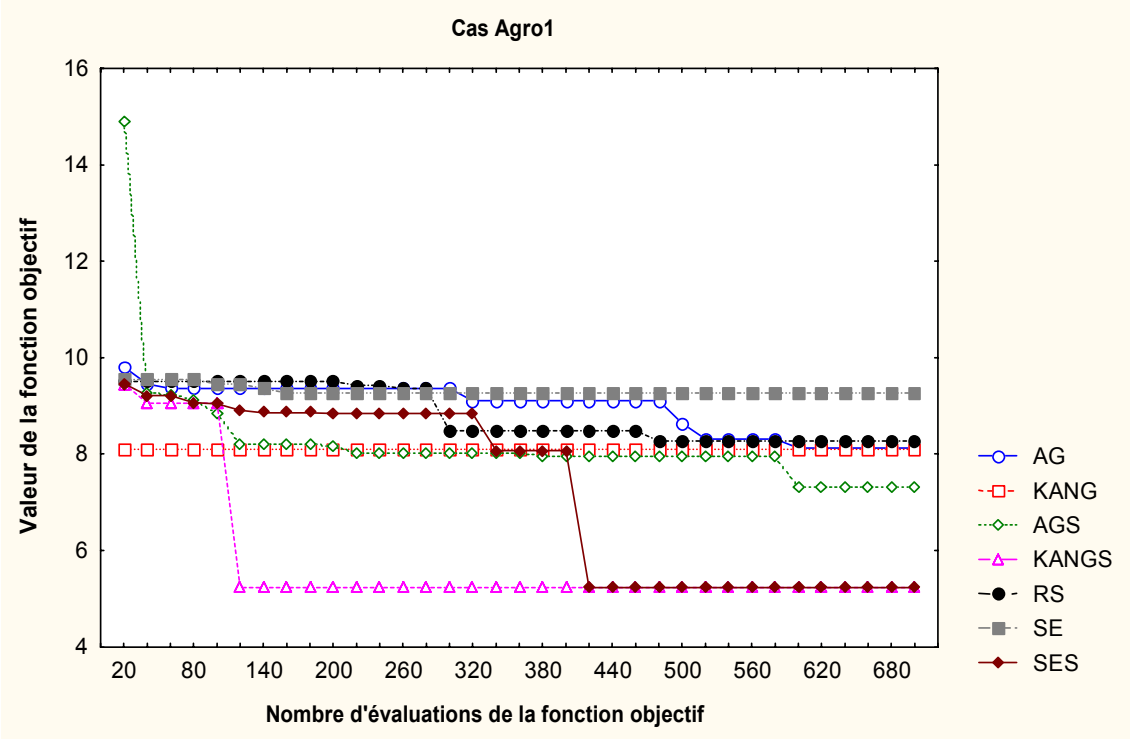


Figure 3.20 (cas Agro1) - *Évolution de la valeur de la fonction objectif, pour 700 évaluations de la fonction objectif.*

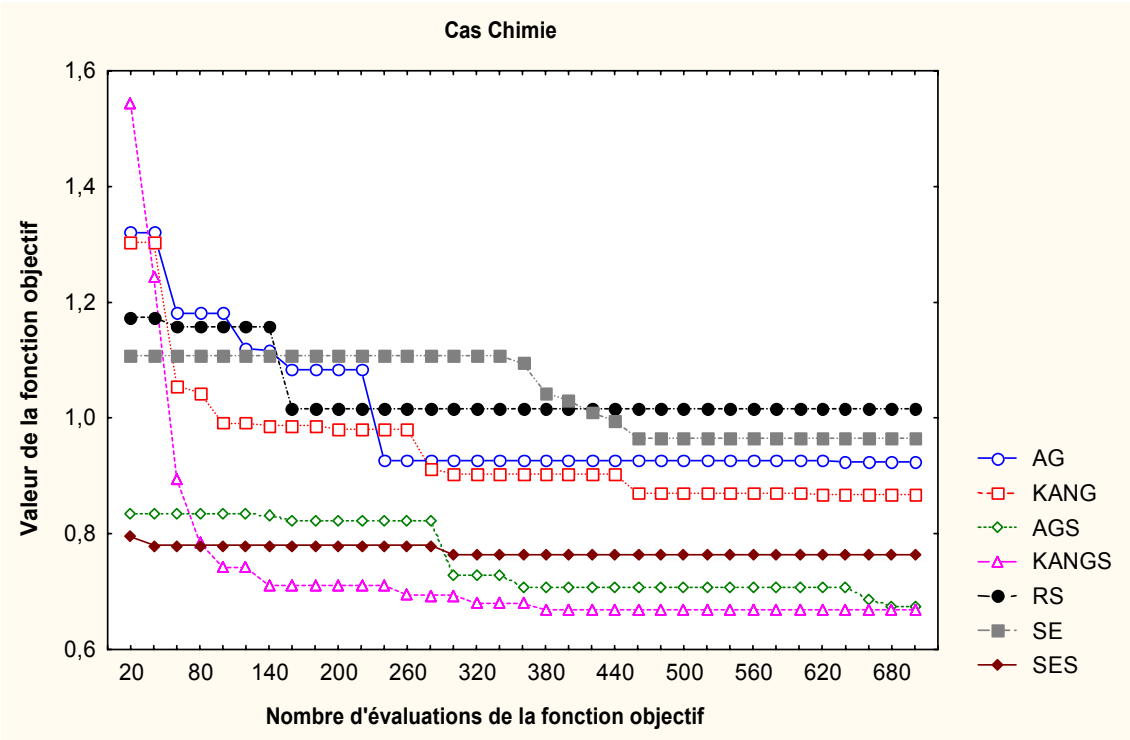


Figure 3.21 (cas Chimie) - *Évolution de la valeur de la fonction objectif, pour 700 évaluations de la fonction objectif.*

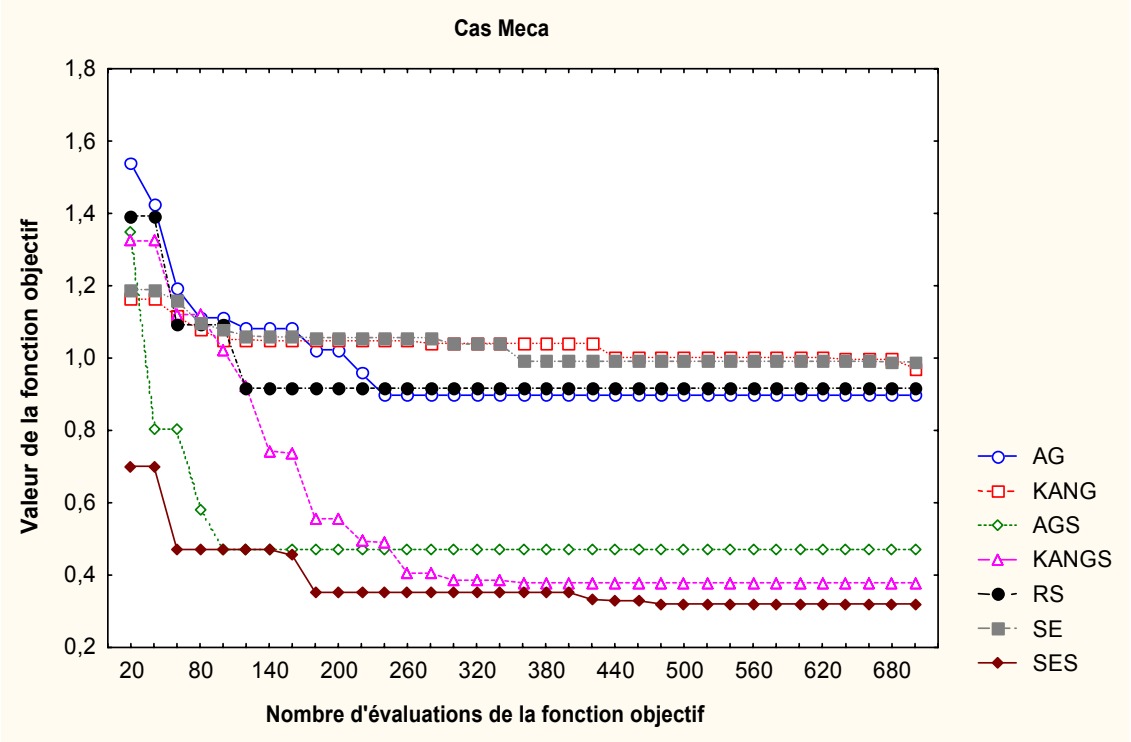


Figure 3.22 (cas Meca) - *Évolution de la valeur de la fonction objectif, pour 700 évaluations de la fonction objectif.*

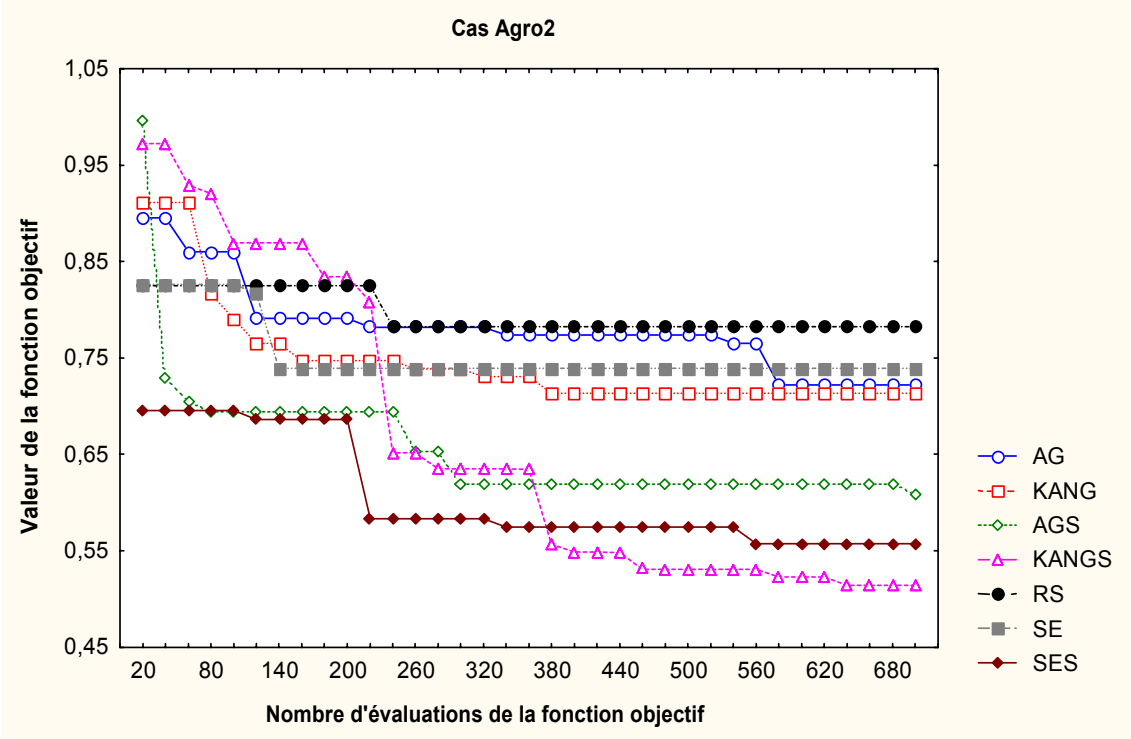


Figure 3.23 (cas Agro2) - *Évolution de la valeur de la fonction objectif, pour 700 évaluations de la fonction objectif.*

	Makespan (jours)	NPS	Durée d'ordo. (secondes)	Temps d'exécution (secondes)
Poids	100%			
Point idéal	0			
Paramétrage manuel	10	8		
RS	8.27	52	16.21	7594
KANG	8.10	52	6.31	4268
GA	8.13	52	6.49	3969
SE	9.28	52	7.36	4376
KANGS	**5.23**	**7**	**3.17**	**2483**
GAS	7.33	12	3.45	3014
SES	**5.23**	**15**	**4.26**	**3692**

Tableau 3.8 (cas Agro1) - *Résultats obtenus après 700 évaluations de la fonction objectif (NPS : nombre de paramètres sélectionnés dans la solution).*

	Retard moyen (jours)	Makespan (jours)	NPS	Durée d'ordo. (secondes)	Temps d'exécution (secondes)
Poids	100%	10%			
Point idéal	0	0			
Paramétrage manuel	79	413	8		
RS	31.96	312	52	3.21	2465
KANG	26.93	307	52	3.07	2403
GA	28.84	307	52	3.48	2557
SE	30.36	297	52	3.05	2467
KANGS	**20.29**	**286**	**8**	**2.31**	**1754**
GAS	**20.49**	**286**	**9**	**2.36**	**1864**
SES	23.52	290	15	2.64	2256

Tableau 3.9 (cas Chimie) - *Résultats obtenus après 700 évaluations de la fonction objectif (NPS : nombre de paramètres sélectionnés dans la solution).*

3.4.3.4. Étude comparative des méthodes d'optimisation

En analysant le tableau 3.6, qui donne les résultats de l'application du test de Mann-Whitney au cas Agro1, on constate que les trois algorithmes AGS, KANGS et SES possèdent des performances moyennes identiques (l'hypothèse nulle ne peut être rejetée avec une confiance de 1%) pour 30 minutes d'exécution. D'un autre côté, pour le cas Chimie, seules les méthodes KANGS et SES possèdent des performances moyennes identiques, la méthode AGS étant moins bonne (voir le tableau 3.7).

A partir de la deuxième série de tests, on peut constater que, dans les cas Agro1, Meca et Agro2, ce sont SES et KANGS qui donnent les meilleurs résultats, alors qu'AGS est un peu moins bonne (voir figures 3.20, 3.21, 3.22 et 3.23). D'un autre côté, pour le cas Chimie, c'est SES qui se comporte moins bien que les deux autres méthodes (AGS et KANGS).

	Taux de service (%)	Makespan (jours)	NPS	Durée d'ordo. (secondes)	Temps d'exécution (secondes)
Poids	100%	10%			
Point idéal	100	0			
Paramétrage manuel	31	618	6		
RS	34.74	618	52	43.56	35419
KANG	39.97	618	52	43.60	30019
GA	35.13	618	52	37.47	28758
SE	38.74	618	52	43.78	29822
KANGS	**45.50**	**618**	**6**	**23.12**	**22209**
GAS	43.66	618	5	20.67	19127
SES	**46.66**	**618**	**5**	**23.31**	**17111**

Tableau 3.10 (cas Meca) - *Résultats obtenus après 700 évaluations de la fonction objectif (NPS : nombre de paramètres sélectionnés dans la solution).*

	Taux de service (%)	Makespan (jours)	Temps de réglage (jours)	NPS	Durée d'ordo. (secondes)	Temps d'exécution (secondes)
Poids	100%	10%	1%			
Point idéal	100	0	0			
Paramétrage manuel	85	262.5	37	11		
RS	88.04	262.5	38.67	52	11.19	8537
KANG	89.26	262.5	37.88	52	11.39	7871
GA	89.11	262.5	37.63	52	10.74	7852
SE	88.80	262.5	36.75	52	11	7863
KANGS	**92.79**	**262.5**	**33**	**7**	**2.73**	**2263**
GAS	91.10	262.5	34.92	10	6.82	3107
SES	**92.02**	**262.5**	**33.21**	**8**	**8.60**	**4264**

Tableau 3.11 (cas Agro2) - *Résultats obtenus après 700 évaluations de la fonction objectif (NPS : nombre de paramètres sélectionnés dans la solution).*

3.5. Conclusion

Les stratégies de sélection de paramètres améliorent significativement les performances des métaheuristiques appliquées. Pour le même temps de calcul, on constate un nombre de solutions visitées plus élevé, et une meilleure valeur de la fonction objectif. De plus, les solutions obtenus en utilisant ces stratégies contiennent très peu de paramètres, ce qui permet d'interpréter plus facilement les résultats et de réduire le temps nécessaire pour effectuer un ordonnancement.

Bien que les résultats illustrés dans ce chapitre soient très prometteurs, l'analyse concerne seulement quatre problèmes tests, et ne permet donc pas de généraliser ces

conclusions à d'autres instances. Seule une étude plus complète, menée sur un grand nombre d'instances issues de cas industriels réels, peut permettre de confirmer la supériorité des approches utilisant la sélection de paramètres, et valider l'intérêt de ces méthodes dans un contexte industriel. Une application réussie de ces approches à un cas réel est aussi présentée en annexe C.

Conclusion et perspectives

1. Conclusions

L'utilisation des logiciels d'ordonnancement industriel fait intervenir une multitude de paramètres, dont le réglage influence fortement la qualité des résultats. A l'heure actuelle, ce réglage est effectué de façon manuelle, au cours de l'installation initiale du logiciel. Plusieurs jours sont alors nécessaires à un expert pour arriver à un réglage acceptable. Malgré ce travail souvent fastidieux, ces paramètres ne sont jamais modifiés, soit parce que leur nombre est trop important pour qu'il soit humainement possible de tous les ajuster manuellement, soit parce que l'expert ne possède pas assez d'expérience dans le réglage de certains d'entre eux. De plus, une fois fixées, les valeurs de ces paramètres sont rarement remises en cause par les utilisateurs, à cause de leur manque d'expérience et du nombre important de paramètres à ajuster.

Dans cette thèse, nous proposons d'automatiser cette tâche de réglage, au moyen d'un environnement basé sur l'optimisation en boîte noire et sur l'utilisation de métaheuristiques. L'optimisation en boîte noire consiste à considérer qu'on ne dispose d'aucune information sur les fonctions (ou la fonction) à optimiser, à l'exception de la possibilité d'obtenir leurs valeurs en un point après avoir effectué une simulation. Dans notre cas, la fonction boîte noire considérée est un logiciel d'ordonnancement, les paramètres de la fonction étant les entrées du logiciel. Pour évaluer cette fonction en un point (ensemble de valeurs des paramètres intervenant en entrée), un ordonnancement est effectué avec le logiciel, des indicateurs de performance étant ensuite récupérés en sortie pour évaluer la qualité de l'ordonnancement. Trois métaheuristiques (algorithme génétique, stratégie d'évolution et méthode du Kangourou) ont été implémentées, et testées sur des problèmes d'ordonnancement issus de cas industriels réels.

Nos expérimentations ont été menées sur un seul type de paramètres utilisés par le logiciel d'ordonnancement *Ortems* : les poids associés à des règles de priorité. Le nombre important de ces paramètres et le temps de calcul élevé nécessaire pour réaliser

un ordonnancement nous ont conduits à introduire des stratégies de sélection de paramètres au sein des métaheuristiques.

Les tests effectués montrent une grande supériorité des métaheuristiques utilisant la sélection, par rapport à leurs versions standard. De plus, les résultats obtenus en appliquant cet environnement au logiciel d'ordonnancement *Ortems* sur des cas industriels réels sont spectaculaires : une amélioration significative, allant de 10 à 50 %, a été obtenue, par rapport aux meilleurs résultats trouvés par les consultants ou les utilisateurs, après plusieurs mois d'utilisation du logiciel. L'annexe C présente une première application réussie, sur un cas industriel réel.

Bien que notre étude montre qu'un tel environnement d'optimisation peut être utilisé avec succès pour améliorer les performances d'un logiciel d'ordonnancement, plusieurs difficultés restent à surmonter :

- *le réglage des paramètres utilisés par les métaheuristiques est une tâche très coûteuse et difficile.* Certains auteurs [Laguna et Adenso-Diaz, 2002] affirment même que, dans le temps total destiné à la conception et à l'expérimentation d'une nouvelle heuristique ou métaheuristique, 10 % est consacré au développement, et 90% au réglage des paramètres d'optimisation.
- *Les logiciels d'ordonnancement commercialisés n'ont pas été conçus pour une utilisation en boîte noire.* Un appel répété de la fonction d'ordonnancement du logiciel peut provoquer des problèmes liés à des « bugs » dans le logiciel : consommation de mémoire croissante, erreurs d'exécution, influence de la situation initiale sur l'ordonnancement, ….
- *Certaines situations pratiques nécessitent des temps d'ordonnancement trop importants* (de l'ordre d'une heure de calcul), alors que les métaheuristiques exigent un nombre d'ordonnancements (ou d'évaluations de la fonction objectif) assez élevé (plusieurs centaines).
- Les clients et consultants ne sont pas familiarisés avec la philosophie des métaheuristiques, qui constituent des approches assez différentes des méthodes d'optimisation classiques :
  - la solution est parfois d'autant meilleure que le temps de calcul est plus important,
  - ces méthodes ne sont pas déterministes (deux exécutions avec les mêmes conditions initiales peuvent donner des résultats différents).
- *Difficulté d'évaluer un ordonnancement.* Très souvent, les indicateurs standard d'*Ortems* ne conviennent pas pour évaluer le problème d'ordonnancement considéré ; il est alors nécessaire d'analyser les préférences du planificateur et de proposer de nouveaux indicateurs spécifiques, ce qui peut être coûteux en temps et en développement.
- *Problème de robustesse de la solution proposée* (voir la partie « perspectives de recherche »).

2. Perspectives de recherche

A l'avenir, plusieurs directions de recherche peuvent être envisagées.

2.1. Le problème de la robustesse du réglage

Pour certains cas industriels, le coût d'un paramétrage peut être trop important en temps de calcul et ressources nécessaires (matérielles et humaines) ; il est alors nécessaire de conserver le même réglage durant une longue période de temps (plusieurs jours ou plusieurs semaines). Au cours de cette période, les données d'ordonnancement peuvent changer de façon significative, pouvant remettre en cause la qualité du réglage choisi. Dans de telles situations, l'« efficacité » du réglage doit être peu sensible à une perturbation dans les données initiales, liée à l'évolution du contexte de l'ordonnancement. Il est alors intéressant de développer des indicateurs pour évaluer cette propriété de robustesse temporelle pour un réglage donné.

2.2. Généralisations

Deux types de généralisations dans l'application d'*Ortems Optimizer* peuvent être envisagées :

- généralisation à d'autres paramètres : jusqu'ici, nos expérimentations ont concerné uniquement deux types de paramètres utilisés par le logiciel *Ortems* : les critères d'ordonnancement et les critères de placement. D'autres paramètres peuvent influencer la qualité d'un ordonnancement : calendriers machines, délais clients, d'autres paramètres internes du moteur d'ordonnancement, …
- généralisation à d'autres logiciels d'ordonnancement : *Ortems Optimizer* est un environnement d'optimisation générique, pouvant s'appliquer à des logiciels d'ordonnancement autres qu'*Ortems*.

2.3. Approche multicritère

L'approche multicritère présentée dans le chapitre 2 présente plusieurs difficultés :

- la détermination des poids intervenant dans le calcul de la fonction objectif est difficile,
- après chaque étape de calcul, une seule solution est présentée au planificateur. D'autres approches multicritères (de type Pareto) peuvent être utilisées pour offrir plus de choix au planificateur.

2.4. Fonctions objectifs coûteuses

Dans le cas où l'ordonnancement est trop coûteux en temps de calcul par rapport au temps disponible pour effectuer l'optimisation, les méthodes d'optimisation proposées au chapitre 2 ne sont pas adaptées. Plusieurs méthodes récentes, spécialement conçues pour les problèmes ayant des fonctions objectifs dont l'évaluation est coûteuse en temps de calcul, peuvent être utilisées. On peut citer :

- la méthodologie des surfaces de réponses [Brekelmans et al., 2001] [Jones et al. 1998],
- les méthodes de recherche par motifs [Serafini, 1998].

Une autre approche consiste à proposer une implémentation parallèle des algorithmes proposés. En utilisant plusieurs ordinateurs en parallèle, on peut réduire considérablement les temps de calcul [Pierreval et Paris, 2000].

2.5. Sélection de paramètres

Si l'on se restreint aux problèmes ayant des paramètres à valeurs binaires (0 ou 1), notre problème de sélection de paramètres se ramène au problème connu dans le domaine de l'apprentissage automatique sous le nom de « sélection de sous-ensemble d'attributs ». Ce problème a fait l'objet de recherches intensives effectuées par les deux communautés de la statistique et de l'apprentissage automatique. Une direction de recherche intéressante consiste à utiliser les différentes approches proposées dans la littérature [Langley, 1994] [Blum et Langley, 1997].

2.6. Relaxation de contraintes

Une autre idée consiste à lancer une optimisation en relaxant certaines contraintes liées à l'ordonnancement, puis à les réintroduire pour chercher une « bonne » solution au voisinage de la solution obtenue avec une relaxation des contraintes. Cette approche est à notre avis particulièrement utile quand le problème d'ordonnancement est très contraint. Les tests effectués jusqu'ici sur certains environnements de production ont donné des résultats prometteurs, mais une expérimentation approfondie reste nécessaire afin de valider ce type d'approche.

2.7. Conclusion

Bien que les directions de recherche qui peuvent être envisagées pour améliorer les méthodes présentées dans cette thèse soient nombreuses nous pensons que le principal défi reste dans la mise en œuvre de ces approches dans un contexte industriel. Un effort de recherche important doit être consacré aux expérimentations numériques et applications aux situations diverses et variées rencontrées en pratique, le but étant de :

- valider la méthodologie présentée ici sur la base d'un ensemble plus important de problèmes d'ordonnancement industriel,
- déterminer de manière plus précise les limites de cette approche (types de problèmes pour lesquels elle n'est pas adaptée, cas où elle est trop coûteuse pour être réalisable dans un contexte industriel…),
- comparer de manière plus pertinente l'avantage d'une telle méthodologie par rapport au paramétrage manuel (basé sur l'expérience).

Annexe A

Liste des indicateurs de performance d'*Ortems*

Pour évaluer les performances d'un planning, on peut distinguer trois catégories d'indicateurs : les indicateurs liés aux machines, les indicateurs liés aux ordres de fabrication et les indicateurs généraux.

1. Indicateurs liés aux machines

Ces indicateurs sont calculés à partir d'indicateurs élémentaires définis sur une seule machine, en utilisant la moyenne arithmétique, le minimum ou le maximum. Par exemple, l'indicateur temps de réglage moyen est calculé en prenant la moyenne arithmétique des temps de réglage par machine. La liste des indicateurs élémentaires (liés aux machines) disponibles dans *Ortems* est la suivante :

- *Taux de Charge :* c'est le rapport entre la somme des durées des tâches (plus les temps de changement d'outils) planifiées sur la machine et l'horizon total de planification.
- *Taux d'Occupation :* c'est le rapport entre la somme des durées des tâches planifiées sur la machine et l'horizon de travail de cette machine. L'horizon de travail de la machine est la différence entre la date de fin de la dernière tâche planifiée sur la machine et la date de début de la première tâche qu'elle produit, dont on soustrait la somme des arrêts du calendrier de la machine dans cette période.
- *Temps de Réglage :* le temps de réglage, exprimé en jours, est la somme des durées de changement d'outils et de changement de valeurs de paramètres effectués sur chaque machine.

- *Temps d'Attente :* le temps d'attente (ou temps mort) d'une machine est l'expression, en jours, de la différence sur l'horizon global de planification entre la durée des tâches planifiées sur cette machine et ses arrêts.

2. Indicateurs liés aux ordres de fabrication

Comme pour les indicateurs liés aux machines, les indicateurs liés aux ordres de fabrication (OF) sont calculés à partir d'indicateurs élémentaires définis sur un seul OF. La liste des indicateurs élémentaires (liés aux OF) disponibles dans *Ortems* est la suivante :

- *Retard :* cet indicateur donne le retard d'un OF par rapport à son délai prévu (délai calculé).
- *Taux de service :* le taux de service est le rapport entre le nombre d'OF en avance et le nombre total d'OF.
- *Retard normalisé :* cet indicateur d'alerte est le rapport entre le retard d'un OF et son chemin critique. Il permet de pondérer les retards des OF en fonction de leur durée théorique minimale.
- *Écoulement :* c'est le rapport (chemin critique / temps de séjour), c'est-à-dire le rapport entre la durée minimale théorique et la durée réelle de planification. Ainsi, un taux d'écoulement faible correspond à un Encours non optimisé, puisque le temps d'attente de l'OF est important.
- *Temps de séjour (ou temps de cycle planifié) :* le temps de séjour correspond à la durée réelle de planification de l'OF, c'est-à-dire : (date de fin de la dernière tâche de l'OF - date de début de la première tâche de l'OF).
- *Chemin critique :* c'est la durée théorique minimale de l'OF. Elle correspond à la somme des durées des tâches constituant le chemin critique de l'OF défini par les différentes contraintes spécifiées dans la gamme associée.
- *Temps d'attente :* c'est la différence entre le temps de séjour et le chemin critique.

3. Indicateurs généraux

- *Taux de juste à temps :* le taux de juste à temps permet de prendre en compte la capacité à terminer un OF le plus rapidement possible (écoulement maximal) et le plus près possible de son délai.
- *Efficacité :* obtenue en multipliant le taux de juste à temps par le taux de charge moyen. Cet indicateur prend en compte la charge des machines, les retards et l'écoulement.
- *Efficacité client :* correspond au produit du taux de service par le taux d'écoulement moyen. La valeur de l'efficacité client est maximale si le taux de service est maximal (tous les OF sont en avance), et si le taux d'écoulement est maximale. Dès que l'un des deux est faible, sa valeur est faible.

Annexe B

Liste des règles de priorité utilisées par *Ortems*

Ortems utilise deux familles de règles élémentaires de priorité : les règles d'ordonnancement et les règles de placement. L'application de chacune de ces règles revient à optimiser une fonction numérique définie sur l'ensemble des tâches à ordonnancer.

1. Les règles d'ordonnancement

Les règles d'ordonnancement sont utilisées pour choisir une tâche, parmi un ensemble de tâches appartenant à un même îlot de machines. Selon que l'ordonnancement soit au plus tôt ou au plus tard, on peut distinguer trois familles de règles élémentaires : spécifiques à l'ordonnancement au plus tôt, spécifiques à l'ordonnancement au plus tard et commun aux deux types d'ordonnancement (au plus tôt et au plus tard). Toutes ces familles seront présentées dans ce qui suit :

1.1. Les règles utilisables pour l'ordonnancement au plus tôt ou au plus tard

(O1) *Priorité maximum de l'OF :* la priorité d'un OF est un nombre entier strictement positif représentant le degré d'urgence de l'OF (plus la valeur de la priorité est élevée, plus l'OF est considéré comme urgent).

(O2) *Degré minimum d'affectation d'une tâche aux machines :* ce degré d'affectation est évalué en calculant le nombre de machines utilisables pour la tâche considérée. La contrainte d'affectation aux machines est d'autant plus forte que ce nombre est faible.

(O3) *Quantité d'une tâche (minimum ou maximum)*,

(O4) *Durée d'une tâche (minimum ou maximum)*,

(O5) *Temps de séjour minimum :* pour une tâche donnée, cette règle consiste à évaluer le maximum des dates de fin des tâches qui la précèdent. La minimisation de cette règle permet de minimiser les temps de séjour des OF. Le temps de séjour d'un OF étant sa durée réelle de planification, c'est-à-dire la différence entre la date de fin de la dernière tâche et la date de début de la première.

(O6) *Rang minimum du groupe de ressources limitées :* l'application de cette règle revient à minimiser la valeur du rang du groupe de ressources limitées utilisé par la tâche considérée.

(O7) *Chemin critique minimum restant de l'OF :* le chemin critique restant d'un OF (pour une tâche donnée) est égale à $f - d$, où f est la date de fin de l'OF (auquel appartient la tâche) et d la date de début de la tâche.

(O8) *Rassembler les Séries :* l'application de cette règle, au cours de l'ordonnancement, consiste à donner plus de priorité aux tâches qui appartiennent à la même Série que celle de la dernière tâche ordonnancée. Une série étant un concept permettant de regrouper un ensemble d'OF tout au long de leur cycle de vie dans *Ortems*. Il peut s'agir d'un regroupement d'OF d'une même commande, entrant dans l'élaboration d'un même produit fini, appartenant à la même campagne ou possédant les mêmes caractéristiques.

(O9) *Temps de changement minimum du paramètre continu :* un paramètre continu modélise une caractéristique d'article prenant ses valeurs dans un ensemble continu (l'épaisseur, le poids, la concentration…). A chaque paramètre continu est associé un sens de variation qui peut être croissant, décroissant ou mixte (voir figure 1 ci-dessous) et un temps de changement. L'application de cette règle consiste à accorder plus de priorité aux tâches qui respectent le sens de variation du paramètre continu et minimisent son temps de changement.

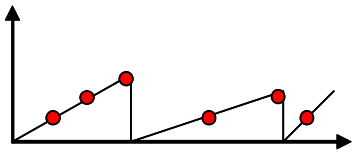
(O10) *Valeur du paramètre continu :* l'application de cette règle consiste à accorder plus de priorité aux tâches dont la valeur corrigée du paramètre continu est minimum. Cette valeur corrigée est égale à la valeur v du paramètre continu si son sens est croissant et $M - v$ s'il est décroissant (M étant un nombre entier suffisamment grand). Dans le cas mixte, cette règle vaut $v - w$, où w est la valeur du paramètre pour la dernière tâche ordonnancée sur la machine à laquelle la tâche considérée a été affectée.

(O11) *Priorité aux tâches compatibles avec un cycle de batch ouvert* (cas où il existe des machines en mode Batch) *:* cette règle permet d'accorder plus de priorité aux tâches qui peuvent s'insérer dans un cycle de batch ouvert.

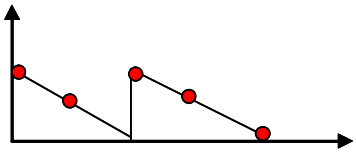
(O12) *Priorité aux tâches dont la durée est proche d'un cycle de batch en cours de remplissage* (cas où il existe des machines en mode Batch) *:* cette règle permet d'homogénéiser les durées des tâches dans les batch, lorsque la dispersion des durées des tâches est grande. Elle consiste à minimiser l'écart entre la durée de la tâche et le temps de cycle d'un batch en cours de remplissage.

1.2. Les règles spécifiques à l'ordonnancement au plus tôt

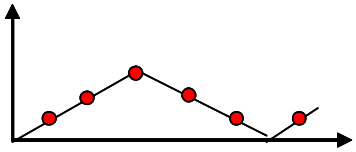
(O13) *Coefficient d'ordonnancement d'OF minimum :* le coefficient d'ordonnancement d'OF, pour une tâche donnée, est égal à $(l-d-c)/c$, où l est le délai de l'OF contenant la tâche, d la date de début de la tâche à l'itération courante de l'ordonnancement et c le chemin critique restant de l'OF (voir la règle élémentaire du chemin critique minimum restant de l'OF). Pour tenir compte de la priorité de l'OF, ce coefficient est multiplié par $1/p$ (si la valeur du coefficient est positif) ou par p (dans le cas où le coefficient est négatif).



Sens croissant



Sens décroissant



Sens mixte

Figure 1 - *Sens de variation des paramètres continus dans Ortems.*

(O14) *Date de début au plus tôt d'une tâche :* cette règle est évaluée en calculant la date de début de la tâche à l'itération courante de l'ordonnancement (en tenant compte du temps de réglage).

(O15) *Date de début au plus tôt d'une tâche, sans prise en compte des temps de réglage,*

(O16) *Marge d'OF minimum :* pour une tâche donnée, la marge d'OF est égale à $l-d$, où l est le délai de l'OF contenant la tâche et d la date de début de la tâche à l'itération courante.

(O17) *Délai d'OF minimum :* le délai d'un OF est égale à la date de fin de fabrication au plus tard de l'OF.

(O18) *Délai de tâche minimum :* le délai d'une tâche est évalué en calculant une date de fin de fabrication au plus tard, qu'on déduit à partir d'un jalonnement (à capacité infinie) au plus tard de l'OF.

(O19) *Coefficient d'ordonnancement minimum :* le coefficient d'ordonnancement d'une tâche est égale à $(l-d)/t$, où *l* est le délai de la tâche (voir la règle précédente), *d* la date de début de la tâche à l'itération courante de l'ordonnancement et *t* sa durée opératoire.

(O20) *Date de début de préparation minimum*,

(O21) *Retard algébrique maximum d'OF :* le retard algébrique d'OF, pour une tâche donnée, est égal à $c+d-l$, où *c* représente le chemin critique restant de l'OF (voir la règle élémentaire du chemin critique minimum restant de l'OF) contenant la tâche, *d* la date de début de la tâche à l'itération courante de l'ordonnancement et *l* est le délai de l'OF.

(O22) *Marge d'admissibilité maximum :* la marge d'admissibilité associée à une tâche permet de mesurer le degré de liberté temporelle (vis-à-vis des dates de fin au plus tard des tâches) dont on dispose quand on commence la séquence par la tâche concernée. Elle est définie comme étant le décalage maximum que l'on pourrait avoir sur la date de début de la tâche concernée si l'on devait respecter les dates de fin au plus tard des autres tâches en conflit à l'itération courante de l'ordonnancement.

(O23) *Marge d'admissibilité délai positive :* l'application de cette règle élémentaire consiste à accorder plus de priorité aux tâches qui ont une marge d'admissibilité positive ou nulle.

(O24) *Date de début d'OF minimum :* l'application de cette règle consiste à donner plus de priorité aux tâches dont les OF ont la plus petite date de début au plus tôt.

(O25) *OF de semi-finis partiellement satisfaits sur stock :* cette règle permet de sélectionner les OF de semi-finis dont la quantité nécessaire est en partie satisfaite par les stocks existants.

(O26) *OF de produits finis satisfaits sur stock :* cette règle permet de sélectionner les OF de produits finis qui sont entièrement fabriqués à partir des semi-finis présents en stock

(O27) *Date de début d'une tâche dont le groupe contraint contient une tâche affectée à un îlot goulot :* cette règle consiste à minimiser la date de début au plus tôt de la première tâche affectée à un îlot goulot et appartenant au même groupe contraint que la tâche considérée.

(O28) *Commencer par les tâches de statut avancé :* les tâches de statut avancé sont celles qui sont en cours de réalisation, mais ne sont pas calées en début de l'horizon de planification.

(O29) *Priorité aux OF non prévisionnels :* cette règle permet de garantir que les tâches des OF fermes seront planifiées avant toutes les tâches des OF prévisionnels.

(O30) *Priorité d'affectation de tâche maximum.*

1.3. Les règles spécifiques à l'ordonnancement au plus tard

(O31) *Date de fin de tâche maximum :* l'application de cette règle consiste à accorder plus de priorité aux tâches dont la date de fin (calculée à l'itération courante de l'ordonnancement, en tenant compte du temps de réglage) est la plus grande.

(O32) *Date de début au plus tard d'une tâche, sans prise en compte des temps de réglage,*

(O33) *Avance minimum :* l'avance d'une tâche est égale à $l - f$, où l est le délai de l'OF contenant la tâche et f la date de fin de la tâche à l'itération courante de l'ordonnancement.

(O34) *Délai d'OF maximum,*

(O35) *OF de semi-finis non partiellement satisfaits sur stock :* cette règle permet de sélectionner les OF de semi-finis dont la quantité nécessaire n'est pas en partie satisfaite par les stocks existants.

(O36) *OF de produits finis non satisfaits sur stock :* cette règle permet de sélectionner les OF de produits finis qui ne sont pas entièrement fabriqués à partir des semi-finis présents en stock

(O37) *Priorité aux tâches dont le statut n'est pas « en cours » :* les tâches en cours sont celles qui sont en cours d'exécution et sont calées au début de l'horizon de planification. L'application de cette règle consiste à ordonnancer en premier les tâches qui ne sont pas en cours, de sorte qu'à l'issue de l'ordonnancement au plus tard, celles-ci se trouvent affectées en tête des machines.

2. Les règles de placement

Ces règles sont utilisées pour choisir, dans un îlot donné, la machine sur laquelle doit passer une certaine tâche.

(P1) *Répartir la charge :* cette règle permet d'équilibrer la charge sur les machines d'un îlot. Elle consiste à accorder plus de priorité à la machine (de la liste de machines utilisables) dont la charge, après placement de la tâche considérée, se rapproche de la charge des autres machines du même îlot.

(P2) *Machine disponible le plus tard (resp. le plus tôt) :* cette règle de placement permet de sélectionner, parmi la liste de machines utilisables, la machine pour laquelle la date de fin de la dernière tâche planifiée est la plus grande (resp. la plus petite).

(P3) *Plus faible nombre de tâches en affectation forcée :* l'application de cette règle revient à sélectionner, dans la liste des machines utilisables, celle dont le nombre de tâches en affectation forcée est le plus faible.

(P4) *Minimiser les temps de réglage :* l'application de cette règle consiste à sélectionner, parmi la liste de machines utilisables, celle pour laquelle la tâche considérée nécessite le plus petit temps de changement.

(P5) *Priorité des cadences la plus petite :* cette règle de placement permet de sélectionner, parmi la liste de machines utilisables, celle dont la priorité est la plus faible.

(P6) *Quantité de la tâche proche de la capacité minimum :* cette règle sert dans le cas de modélisation de cuves avec contrainte de remplissage. Elle permet de sélectionner, parmi la liste de machines utilisables, celle dont la capacité minimale est la plus proche par excès de la quantité de la tâche.

(P7) *Quantité de la tâche proche de la capacité maximum :* cette règle sert dans le cas de modélisation de cuves avec contrainte de remplissage. Elle permet de sélectionner, parmi la liste de machines utilisables, celle dont la capacité maximale est la plus proche par défaut de la quantité de la tâche.

(P8) *Quantité de la tâche entre la capacité minimum et maximum :* cette règle sert dans le cas de modélisation de cuves avec contrainte de remplissage. Elle permet de sélectionner, parmi la liste de machines utilisables, celle dont les capacités minimales et maximales bornent la quantité de la tâche.

(P9) *Priorité de placement sur les cycles de batch ouvert :* l'objectif de cette règle est de compléter en priorité les cycles de batch ouverts, afin d'optimiser leurs remplissages.

(P10) *Cycle le plus proche :* l'objectif de cette tâche est d'homogénéiser les durées des tâches contenues dans les cycles de batch. Elle consiste à choisir la machine batch dont le dernier cycle ouvert, dans lequel on peut placer la tâche, a la durée la plus proche de celle de la tâche.

(P11) *Minimiser le temps de transfert entre zones :* cette règle de placement permet de sélectionner, parmi la liste de machines utilisables, celle qui se trouve dans la même zone que les machines des prédécesseurs de la tâche (ou dans la zone la plus proche du point de vue du temps de transfert).

(P12) *Moins d'affectation par défaut :* cette règle contribue à répartir la charge machine. Elle consiste à accorder plus de priorité aux machines ayant le minimum de tâches affectées, parmi celles qui sont en conflit à l'itération courante de l'ordonnancement.

(P13) *Ressources limitées :* cette règle de placement permet de sélectionner, parmi la liste de machines utilisables, celle qui respecte les contraintes de ressources limitées et groupe de ressources.

(P14) *Rassembler les séries :* cette règle de placement permet de sélectionner, parmi la liste de machines utilisables, celle dont la série de la dernière tâche ordonnancée est la même que la tâche à placer.

(P15) *Respecter la fin au plus tard de la tâche :* cette règle de placement permet de sélectionner, parmi la liste de machines utilisables, celle qui respecte la date de fin au plus tard de la tâche à planifier. Cette date est calculée à partir d'un jalonnement à capacité infinie au plus tard de l'OF auquel la tâche appartient.

(P16) *Temps de changement de paramètre continu minimum :* cette règle de placement permet de sélectionner, parmi la liste de machines utilisables, celle qui induit le plus faible temps de changement lié au paramètre continu spécifié, tout en respectant le sens de variation des valeurs de ce paramètre.

(P17) *Valeur de paramètre continu minimum :* cette règle de placement permet de sélectionner, parmi la liste de machines utilisables, celle qui induit la valeur de paramètre la plus proche de celle précédemment ordonnancée, tout en respectant le sens de variation des valeurs de ce paramètre.

Annexe C

Première « success story »

Communiqué de presse

Le groupe GIVAUDAN renouvelle sa confiance à FINMATICA en implémentant la solution FINeCHAIN OPTIMIZER, et annonce une augmentation relative du taux de service de 50 % et une réduction du temps de séjour des produits de 30 %.

GIVAUDAN a implémenté la solution FINeCHAIN OPTIMIZER, basée sur des algorithmes évolutifs et une technologie web, visant à optimiser les résultats obtenus par la solution FINeCHAIN Production Scheduler installée au sein du groupe depuis 1996.
GIVAUDAN annonce des résultats significatifs en matière de taux de service et de temps de séjour des produits.
La solution FINeCHAIN Production Scheduler est parfaitement intégrée à l'ERP de GIVAUDAN, grâce à la solution FINeCHAIN VIC.

Paris, France – 26 février 2004: Finmatica a annoncé aujourd'hui que le groupe GIVAUDAN enregistre des résultats significatifs après l'installation de la solution FINeCHAIN OPTIMIZER de FINMATICA.

Le groupe GIVAUDAN, basé en Suisse et leader mondial en fabrication de parfums et arômes, a engagé « une Démarche de Recherche de l'Excellence Industrielle ». Dans un contexte d'exigence croissante de réactivité et de réduction des délais, GIVAUDAN a souhaité atteindre une meilleure réactivité, améliorer les délais et réduire les temps de cycle pour faire face à la demande fluctuante de ses clients.

« La fabrication chimique des ingrédients de parfumerie se fait selon des procédés divers mis en oeuvre dans des unités de production très variées. FINeCHAIN Production Scheduler est l'outil de choix qui nous permet de réaliser l'ordonnancement de ces fabrications complexes. Il offre à nos planificateurs une vision globale et précise de la capacité et des délais. Cette maîtrise du planning, associée à celle des matières premières et de la main d'oeuvre, permet une gestion optimale de la production, déclare Monsieur SCHAERER, Directeur de la production chimique.

GIVAUDAN, client FINMATICA depuis 1996, planifie et ordonnance à capacité finie et en tenant compte des contraintes liées à son activité, les ateliers de fabrication des ingrédients de parfumerie, à partir du module FINeCHAIN Production Scheduler de la suite FINeCHAIN.

« FINeCHAIN est l'outil idéal pour connaître au mieux l'occupation des postes de charge. Il permet un meilleur respect des délais tout en optimisant cette occupation », déclare Monsieur MARTIN, Responsable Ordonnancement.

Le processus d'amélioration continue, engagé par GIVAUDAN, a conduit au déploiement de la fonctionnalité d'optimisation des gammes alternatives et des moteurs d'optimisation avancée de FINeCHAIN Optimizer. Basé sur les techniques innovantes d'algorithmes évolutifs et sur l'architecture Web de FINeCHAIN, Optimizer permet à GIVAUDAN d'accroître son efficacité, **en optimisant les résultats obtenus par l'ordonnancement effectué par FINeCHAIN Production Scheduler, le logiciel de planification et d'ordonnancement de la production de FINMATICA.**

« Nous avons constaté des résultats significatifs : une augmentation relative du taux de service client de 50%, une réduction du temps de séjour moyen des produits de 30%, et une réduction importante des encours de production », ajoute Monsieur SCHAERER. « Ces résultats sont en parfaite adéquation avec les objectifs que nous nous étions fixés dans le cadre de notre démarche de recherche de l'excellence industrielle ».

Le logiciel est parfaitement intégré au système d'information du groupe (BPCS), grâce au module FINeCHAIN VIC (outil de type EAI), développé par FINMATICA. Les équipes GIVAUDAN ont été formées à la technologie VIC.

«La connaissance de l'outil EAI nous a permis de réaliser en un temps très court, l'intégration avec BPCS de façon très satisfaisante », déclare Monsieur MARCHAND, Responsable du Support Technique.

« GIVAUDAN est l'exemple d'un partenariat durable basé sur la création de valeur. Il s'appuie sur les compétences de nos équipes et de nos produits pour continuellement améliorer les processus de gestion de la chaîne logistique, accroître le niveau de service clients, et accompagner sa Démarche de Recherche de l'Excellence Industrielle », ajoute René DESVIGNES Responsable de la Division Supply Chain Management de FINMATICA.

***

A propos de GIVAUDAN

Résultat de sa longue expérience, Givaudan regroupe l'expérience de différentes sociétés depuis la fin du 18ème siècle jusqu'aux plus récentes connaissances de l'industrie des parfums et saveurs. GIVAUDAN utilise ses fortes capacités de recherche et développement produits, incluant l'une des plus anciennes écoles de parfumerie, au service de sa clientèle nombreuse et distinguée, en créant des composants de parfums et saveurs pour une grande variété de produits. Aujourd'hui, GIVAUDAN est une société scientifique, moderne et innovante, avec une couverture et des ressources globales, fortement positionnée pour réussir dans le 21ème siècle.

A propos de FINeCHAIN Optimizer

FINeCHAIN Optimizer est une solution permettant d'améliorer les performances de l'ordonnancement en intervenant au niveau du paramétrage du logiciel d'ordonnancement de FINeCHAIN Production Scheduler. Plusieurs algorithmes avancés d'optimisation (stratégies évolutionnistes, algorithme génétique, algorithme du kangourou) sont utilisés pour automatiser le réglage des paramètres intervenant en entrée du logiciel et en ajustant ces paramètres en fonction des objectifs de performances à atteindre par l'ordonnancement.

FINeCHAIN Optimizer permet de suivre l'évolution de l'optimisation grâce à plusieurs vues graphiques.

A propos de FINMATICA SCM

Finmatica SCM est la division Supply Chain Management du groupe FINMATICA, qui développe, distribue et implémente des applications spécialistes en matière de gestion de la chaîne logistique étendue. Avec plus de 500 clients au niveau international, FINeCHAIN permet d'optimiser les processus liés à la chaîne logistique, en apportant de nombreuses améliorations quant à la gestion des commandes client, à la planification de la chaîne logistique et à la gestion des relations avec les fournisseurs. L'architecture modulaire de FINeCHAIN permet une intégration rapide avec les systèmes d'information existants en assurant un retour sur investissement garanti. Pour plus d'informations, consultez www.finechain.com

A propos de Finmatica

FINMATICA (www.finmatica.com) dont le siège est basé à Milan, développe et distribue des applications dans les domaines du Supply Chain Management, de la Finance, de la Sécurité, et de la gestion documentaire. Finmatica est présent au niveau international avec 22 filiales basées dans 12 pays. Finmatica compte 900 d'employés.

Press contact:

Finmatica France
Valérie GOULEVITCH
Tel: +33.4.37.40.11.07
Email: v.goulevitch@finmatica.com

Références bibliographiques

[Aarts et Korst, 1989] Aarts, E. H. L., and Korst, J. (1989). Simulated annealing and Boltzman Machines: A stochastic approach to combinatorial and neural computing. New York, John Wiley and Sons.

[Al Kazzaz, 1989] Al Kazzaz, Y. (1989). Sur l'ordonnancement d'atelier de fabrication : Approche hiérarchisée et fonctionnement en boucles de pilotage. *Thèse de doctorat de l'Institut National Polytechnique de Grenoble*.

[Al-Sultan et Al-Fawzan, 1997] Al-Sultan, K. S., and Al-Fawzan, M. A. (1997). A tabu search Hooke and Jeeves algorithm for unconstrained optimization. *European Journal of Operational Research*, 103, pp. 198-208.

[Amitay, 2003] Amitay, I. (2003). Direct-search methods and DACE. *Seminar Report*, Department of Aerospace Engineering, Indian Institute of Technology, Bombay, may 2003.

[Andersson, 2000] Andersson J. (2000). A Survey of Multiobjective Optimization in Engineering Design. *Technical report LiTH-IKP-R-1097*, Department of Mechanical Engineering, Linköping University, Linköping, Sweden.

[April et al., 2003] April, J., Glover, F., Kelly, J. P. and Laguna, M. (2003). Practical introduction to simulation optimization. *Proceedings of the 2003 Winter Simulation Conference*, S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

[Archetti et Schoen, 1984] Archetti, F., and Schoen, F. (1984). A survey on global optimization problems: General theory and computational approaches. *Annals of Operations Research*, 1, pp. 87-110.

[Atmar, 1976] Atmar, J. W. (1976). Speculation on the evolution of intelligence and its possible realization in machine form. *Phd thesis*, New Mexico State University, Las Cruces, NM.

[Bäck et al., 1993] Bäck, T., Rudolph, G., and Schwefel, H.-P. (1993). Evolutionary programming and evolution strategies: Similarities and differences. In D. B. Fogel and J. W. Atmar, editors, *Proceedings of the Second Annual Conference on Evolutionary Programming*, Evolutionary Programming Society, La Jolla, CA, pp.11-22.

[Baek et al., 1998] Baek, D. H., Yoon, W. C., and Park, S. C. (1998). A spatial rule adaptation procedure for reliable production control in a wafer fabrication system. *International Journal of Production Research*, 36 (6), 1475-1491.

[Baker, 1984] Baker, K. R. (1984). Sequencing rules and due-date assignments in a job shop. *Management Science*, 30(9), 1093-1104.

[Bandler, 1969] Bandler, J. W. (1969). Optimization methods for computer-aided design. *IEEE Transactions on Microwave Theory and Techniques*, 17 (8), pp. 533-552.

[Barr et al., 1995] Barr, R. S., Golden, B. L., Kelly, J. P., Resende, M. G. C., and Stewart, W. R. (1995). Designing and reporting computational experiments with heuristic methods. *Journal of Heuristics*, 1 (1), pp. 9-32.

[Barr et al., 2001] Barr, R. S., Golden, B. L., Kelly, J., Stewart, W. R., and Resende, M. G. C. (2001). Guidelines for designing and reporting on computational experiments with heuristic methods. *ORSA Journal on Computing*, 5 (1), pp. 2-18 (http://www.research.att.com/~mgcr/doc/ guidelines.pdf).

[Barthelemy et Haftka, 1993] Barthelemy, J.-F. M., and Haftka, R. T. (1993). Approximation concepts for optimum structural design – A review. *Structural optimization*, 5, pp. 129-144.

[Barton, 1998] Barton, R. R. (1998). Simulation metamodels. *Proceedings of the 1998 Winter Simulation Conference*, D. J. Medeiros, E. F. Watson, J. S. Carson and M. S. Manivannan, eds., pp. 167-174.

[Barton, 1993] Barton, R. R. (1993). New tools for simulation metamodels. IMSE Working Paper 93-110, Department of Industrial and Management Systems Engineering, The Pennsylvania State University, University Park, Pennsylvania.

[Battiti et Tecchiolli, 1994] Battiti, R., and Tecchiolli, G. (1994). The reactive tabu search. *In ORSA Journal on Computing*, 6 (2), pp. 126-140.

[Battiti et Tecchiolli, 1995] Battiti, R., and Tecchiolli, G. (1995). Local search with memory: Benchmarking RTS. *In Operations Research Spektrum*, 17 (2/3), pp. 67-86.

[Becker et Szczerbicka, 1998] Becker, M. and Szczerbicka, H. (1998). Modeling and optimization of Kanban controlle manufacturing systems with GSPN including QN. *In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, San Diego, USA.

[Belanche, 1999] Belanche, LI. A., (1999). A study in function optimization with the Breeder Genetic Algorithm. *LSI Research Report LSI-99-36-R*, Universitat Politècnica de Catalunya.

[Bellman, 1961] Bellman, R. (1961). Adaptive Control Processes. Princeton University Press.

[Ben-Arieh, 1988] Ben-Arieh, D. (1988). Knowledge-based routing and control system for FMS. In S. T. Kumara, A. L. Soyster, R. L. Kashyap (Eds), *Artificial intelligence : manufacturing theory and practice*, Norcross, pp. 631-646.

[Bensana et al., 1988] Bensana, E., Bel, G., and Dubois, D. (1988). OPAL: A multi-knowledge-based system for industrial job-shop scheduling, *International Journal of Production Research*, 26 (5), pp. 795-819.

[Beveridge et al., 1996] Beveridge, J., Graves, C., and Lesher, C. E. (1996). Local search as tool for horizon line matching. *Tech. rep. CS-96-109*, Colorado State University.

[Birattari et al., 2002] Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics, In W. B. Langdon et al. (Eds.), *GECCO 2002*, Morgan Kaufmann, San Francisco, CA, USA, pp. 11-18.

[Blackstone et al., 1982] Blackstone, J. H., Phillips, D. T., and Hogg, G. L. (1982). A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *International Journal of Production Research*, 20(1), 27-45.

Blum, A. L., and Langley, P. (1997). Selection of relevant features and examples in machine learning. In R. Greiner and D. Subramanian, eds., *Artificial Intelligence on Relevance*, 97, Artificial Intelligence, pp. 245-271.

[Blum et Roli, 2001] Blum, C., and Roli, A., (2001). Metaheuristics in combinatorial optimization : overview and conceptual comparison. *Technical Report No TR/IRIDIA/2001-13*, Université libre de Bruxelles.

[Bosman et Thierens, 2000] Bosman, P. A. N., and Thierens, D. (2000). Expanding from discrete to continuous estimation of distribution algorithms: The IDEA. *In Parallel Problem Solving from Nature – PPSN VI*, Springer-Verlag, pp. 767-776.

[Box et Wilson, 1951] Box, G. E. P., and Wilson, K. B. (1951). On the experimental attainment of optimum conditions. *Journal of Royal Statistical Society*, Series B 13 (1), pp. 1-38.

[Brekelmans et al., 2001] Brekelmans, R., Driessen, L., Hamers, H., and Hertog, D. (2001). A new sequential approach to product and process design involving expensive simulations. *Third ASMO UK / ISSMO Conference on Engineering Design Optimization*.

[Bruns, 1993] Bruns, R. (1993). Direct chromosome representation and advanced genetic algorithms for production scheduling. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, Morgan Kaufmann, pp. 352-359.

[Burgin, 1968] Burgin, G. H. (1968). On playing two-person zero-sum games against non-minimax players. *IEEE Transactions on Systems, Man, and Cybenetics*, SSC-5(4), pp. 369-370.

[Burgin, 1973] Burgin, G. H. (1973). Systems identification by quasilinear and evolutionary programming. *Journal of Cybernetics*, 3(2), pp. 56-75.

[Buzacott et Yao, 1986] Buzacott, J. A., and Yao, D. D. (1986). Flexible Manufacturing Systems : a review of analytic models. *Management Science*, 32(7), pp. 890-905.

[Byrne et Taguchi, 1987] Byrne, D. M., and Taguchi, S. (1987). The Taguchi approach to parameter design. *Quality Progress*, December, pp. 19-26.

[Cardoso et al., 1996] Cardoso, M. F., Salcedo, R. L., and Azevedo, S. F. de (1996). The simplex simulated annealing approach to continuous non linear optimization. *Comput. Chem. Eng.*, 20, pp. 1065-1080.

[Cardoso et al., 1997] Cardoso, M. F., Salcedo, R. L., Azevedo, S. F. de, and Barbosa, D. (1997). A simulated annealing approach to the solution of minlp problems. *Comput. Chem. Eng.*, 21, pp. 1349-1364.

[Carson et Maria, 1997] Carson, Y., and Maria, A. (1997). Simulation optimization: methods and applications. *Proceedings of the 1997 Winter Simulation Conference*, pp. 118-126.

[Cerny, 1985] Cerny, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45, pp. 41-51.

[Chandra et Talavage, 1991] Chandra, J., and Talavage, J. (1991). Intelligent dispatching for flexible manufacturing. *International Journal of Production Research*, 29, pp. 2259-2278.

[Chelouah et Siarry, 2000] Chelouah, R., and Siarry, P. (2000). Tabu search applied to global optimization. *European Journal of Operational Research*, 123, pp. 256-270.

[Chelouah et Siarry, 2004] Chelouah, R., and Siarry, P. (2004). A hybrid method combining continuous tabu search and Nelder-Mead simplex algorithms for the global optimization of multiminima functions. *To appear in European Journal of Operational Research*.

[Chen, 1999] Chen, S. (1999). Is the common good? A new perspective developed in genetic algorithms. *Phd thesis*, Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania.

[Chiadamrong, 2003] Chiadamrong, N. (2003). A sequential procedure for manufacturing system design. *ScienceAsia*, 29, pp. 57-65.

[Choi, 2000] Choi, C. W. (2000). An Overview of Solving Job Shop Scheduling Problem by Local Search Techniques, url = ”citeseer.ist.psu.edu/424201.html” (file: an-overview-of-solving.pdf).

[Collette et Siarry, 2002] Collette, Y. et Siarry, P. (2002). Optimisation multiobjectif. Editions Eyrolles.

[Coy et al., 2001] Coy, S. P., Golden, B. L., Runger, G. C., and Wasil, E. A. (2001). Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, 7 (1), pp. 77-97.

[Cressie, 1993] Cressie, N. A. C. (1993). Statistics for spatial data. Wiley, New York.

[Crowder et al., 1978] Crowder, H. P., Dembo, R. S., and Mulvey, J. M. (1978). Reporting computational experiments in mathematical programming. *Mathematical Programming*, 15, pp. 316-329.

[Cung et al., 2002] Cung, V.-D., Martins, S. L., Ribeiro, C. C. and Roucairol, C. (2002). Strategies for the parallel implementation of metaheuristics. In C. C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, Kluwer, pp. 263-308.

[Cvijovic et Klinowski, 1995] Cvijovic, D., and Klinowski, J. (1995). Taboo search: An approach to the multiple minima problem. *Science 667*, pp. 664-666.

[Cvijovic et Klinowski, 2002] Cvijovic, D., and Klinowski, J. (2002). Taboo search: An approach to the multiple minima problem for continuous functions. In P. M. Pardalos and H. E. Romeijn (Eds.), *Handbook of Global Optimization*, Kluwer Academic Publishers, Boston, MA, pp. 387-406.

[Davis, 1991] Davis, L. (1991). The handbook of genetic algorithms, Von Nostrand Reinhold, New York.

[Davis et Fox, 1994] Davis, G., and Fox, M. (1994). ODO: A constraint-based architecture for representing and reasoning about scheduling

problems. *In Proceedings of the third Industrial Engineering research Conference*.

[Dawkins, 1976] Dawkins, R. (1976). The selfish gene. Oxford University Press.

[Deneubourg et al., 1990] Deneubourg, J.-L., Aron, S., Goss, S., and Pasteels, J.-M. (1990). The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behaviour*, 3, pp. 159-168.

[Devore, 1991] Devore, J. L. (1991). Probability and statistics for engineering and the sciences. Third Edition, Brooks/Cole Publishing Company, Pacific Grove, California.

[Dixon, 1978] Dixon, L. C. W. (1978). Global optima without convexity. *Technical Report*, Numerical Optimization Centre, Hatfield Polytechnic, Hatfield, England.

[Dixon et Szegö, 1978] Dixon, L. C. W., and Szegö, G. P. (1978). The global optimization: An introduction. In Dixon and Szegö, eds., *Towards Global Optimization*, 2, North-Holland, Amsterdam, pp. 1-15.

[Dolan et al., 2000] Dolan, E. D., Lewis, R. M., and Torczon, V. (2000). On the local convergence of pattern search. *ICASE Report No. 2000-36*, Institute for Computer Applications in science and Engineering, NASA Langley Research Center, Hampton.

[Dorigo, 1992] Dorigo, M. (1992). Optimization, learning and natural algorithms (in Italian). *Phd thesis*, DEI, Politecnico di Milano, Italy.

[Dorigo et al., 1996] Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics – Part B*, 26 (1), pp. 29-41.

[Dorigo et al., 1999] Dorigo, M., Di Caro, G., and Cambardella, L. M. (1999). Ant algorithms for discrete optimization. *Art. Life*, 5 (2), pp. 137-172.

[Dorndorf et Pesch, 1995] Dorndorf, U. and Pesch, E. (1995). Evolution based learning in a job shop scheduling environment. *Computers and Operations Research*, 22 (1), pp. 25-40.

[Dréo et al., 2003] Dréo, J., Pétrowski, A., Siarry, P., and Taillard, É. (2003). Métaheuristiques pour l'optimisation difficile. Éditions Eyrolles, Paris.

[Dyn et al., 1986] Dyn, N., Levin, D. and Rippa, S. (1986). Numerical procedures for surface fitting of scattered data by radial basis functions. *SIAM Journal of Scientific and Statistical Computing*, 7(2), pp. 639-659.

[Eiben et Smith, 2003] Eiben, A. E. and Smih, J. E. (2003). Introduction to Evolutionary Computing, Springer.

[Emery et al., 1966] Emery, F. E., O'Hagen, M., and Nolte, S. D. (1966). Optimal design of matching networks for microwave transistor amplifiers. *IEEE Transactions on Microwave Theory and Techniques*, 14, pp. 696-698.

[Engelund et al., 1993] Engelund, W. C., Douglas, O. S., Lepsch, R. A., McMillian, M. M., and Unal, R. (1993). Aerodynamic configuration design using response surface methodology analysis. *AIAA Aircraft Design, Sys. & Oper. Mngmt.*, Monterey, CA, pp. 93-3967.

[Eshelman et Schaffer, 1993] Eshelman, L. J., and Schaffer, J. D. (1993). Real coded genetic algorithms and interval-schemata. *In foundations of genetic algorithms*, 2, Morgan Kaufman Publishers, san Mateo, pp. 187-202.

[ESPRIT 6805, 1995] ESPRIT 6805 (1995). Complan: Concurrent Manufacturing Planning and Shop Control for Small Batch Production. *Public Domain report*, May.

[Fang et al., 1994] Fang, H., Ross, P., and Corne, D. (1994). A promising hybrid GA/heuristic approach for open-shop scheduling problems. *ECAI 94, 11th European Conference on AI*, John Wiley & Sons, New York.

[Fang et Wang, 1994] Fang, K.-T., and Wang, Y. (1994). Number-theoretic methods in statistics. Chapman & Hall, New York.

[Feo et Resende, 1989] Feo, T. A., and Resende, M. G. C. (1989). A probabilistic heuristic for computationally difficult set covering problem. *Operations Research Letters*, 8, pp. 67-71.

[Feo et Resende, 1995] Feo, T. A., and Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6, pp. 109-133.

[Festa et Resende, 2002] Festa, P., and Resende, M. G. C. (2002). GRASP: An annotated bibliography. In C. C. Ribeiro and P. Hansen, editors, *Essays and surveys in Metaheuristics*, pp. 325-367, Kluwer Academic Publishers.

[Fleurent et Ferland, 1994] Fleurent, C., and Ferland, J. A. (1994). Genetic hybrids for the quadratic assignment problem. In Pardalos, P. & Wolkowicz, H., (eds.), Quadratic Assignment and Related Problems, vol. 16, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pp. 173-187.

[Fleury, 1993] Fleury, G. (1993). Méthodes stochastiques et déterministes pour les problèmes NP-Difficiles, *Thèse de doctorat*, Université de Clermont-Ferrand II.

[Fogel, 1962] Fogel, L. J. (1962). Toward inductive inference automata. In proceedings of the International Federation for Information Processing Congress, pp. 395-399, Munich.

[Fogel et al., 1966] Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). Artificial Intelligence through Simulated Evolution. Wiley, New York.

[Fourer et al., 1993] Fourer, R., Gay D., and Kernighan, B. (1993). AMPL: A modeling language for Mathematical Programming. Duxbury Press.

[Franze et Speciale, 2001] Franze, F., and Speciale, N. (2001). A tabu search based algorithm for continuous multiminima problems. *International Journal for Numerical Engineering*, 50, pp. 665-680.

[Friedman, 1991] Friedman, J. H. (1991). Multivariate adaptive regression splines. *The Annals of Statistics*, 19 (1), pp. 1-141.

[Galindo-Legaria, 1994] Galindo-Legaria, C., Pellenkoft, A., and Kersten, M. (1994). Fast, randomized join-order selection – why use transformations? *In Proc. VLDB*, pp. 85-95.

[Garey et Johnson, 1979] Garey, M. R., and Johnson, D. S. (1979). Computers and Intractability: A guide to the theory of NP-Completeness. San Francisco: W. H. Freeman & Co.

[German et German, 1984] German, S., and German, D. (1984). Stochastic relaxation, gibbs distribution and the bayesian restoration in images. *IEEE Trans. Patt. Anan. Mach. Int.*, 6, pp. 721-741.

[Glover, 1986] Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Comp. Oper. Res.*, 13, pp. 533-549.

[Glover, 1994] Glover, F. (1994). Genetic algorithms and scatter search: Unsuspected potentials. *Statistics and Computing*, 4, pp. 131-140.

[Glover et Laguna, 1997] Glover, F., and Laguna, M. (1997). Tabu search. Kluwer Academic Publishers.

[Glover et al., 2000] Glover, F., Laguna, M., and Marti, R. (2000). Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39, pp. 653-684.

[Goldberg, 1989] Goldberg, D. E. (1989). Genetic algorithms in search, optimization and machine learning. Addison Wesley, Reading, MA.

[Goldberg, 1991] Goldberg, D. E. (1991). Real-coded genetic algorithms, virtual alphabets and blocking. *Complex Systems*, 5, pp. 139-167.

[Grabot, 1998] Grabot, B. (1998). Objective satisfaction assessment using neural nets for balancing multiple objectives. *Int. J. Prod. Res.*, 36 (9), pp. 2377-2395.

[Grabot et Geneste, 1994] Grabot, B., and Geneste, L. (1994). Dispatching rules in scheduling: a fuzzy approach. *International Journal of Production Research*, 32 (4), pp. 903-915.

[Graves, 1981] Graves, S. C. (1981). A review of production scheduling. Operations Research, vol. 29, no. 4, pp. 646-675.

[Greenberg, 1990] Greenberg, H. (1990). Computational testing: Why, how and how much. *ORSA journal on Computing*, 2, pp. 7-11.

[Grossman et Davidor, 1992] Grossman, T., and Davidor, Y. (1992). An investigation of a genetic algorithm in continuous parameter space. The Weizmann Institute of science, Rehovot, Israel.

[Haimes et al., 1971] Haimes, Y., Lasdon, L., and Wismer, D. (1971). On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, 1, pp. 296-297.

[Hansen, 1986] Hansen, P. (1986). The steepest ascent mildest descent heuristic for combinatorial programming. *In Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy.

[Hardy, 1971] Hardy, R. L. (1971). Multiquadratic equations of topography and other irregular surfaces. *J. Geophys. Res.*, 76, pp. 1905-1915.

[Hart et Ross, 1998] Hart, E. and Ross, P. (1998). A heuristic combination method for solving job-shop scheduling problems. Proceedings *of the V Parallel Problem Solving From Nature (PPSN V)*, LNCS, vol. 1498, Springer, pp. 845-854.

[Hedar et Fukushima, 2002] Hedar, A., and Fukushima, M. (2002). Hybrid simulated annealing and direct search method for nonlinear unconstrained global optimization. *Optimization Methods and Software*, 17, pp. 891-912.

[Hedar et Fukushima, 2003] Hedar, A.-R., and Fukushima, M. (2003). Tabu search by direct search methods for nonlinear global optimization. *Technical Report 2003-007*, Department of Applied Mathematics & Physics, Kyoto University.

[Hedar et Fukushima, 2004] Hedar, A.-R., and Fukushima, M. (2004). Heuristic pattern search and its hybridization with simulated annealing for nonlinear global optimization. *To appear in Optimization Methods and Software*.

[Hetimansperger, 1984] Hetimansperger, T. P. (1984). Statistical inference based on ranks. John Wiley Sons, Inc.

[Higuchi et al., 2000] Higuchi, T., Tsutsui, S., and Yamamura, M. (2000). Theoretical analysis of simplex crossover for real-coded Genetic Algorithms. *Proc. Of the sixth International Conference on Parallel Problem Solving from nature (PPSN VI)*, pp. 365-374.

[Hoaglin et Andrews, 1975] Hoaglin, D. C., and Andrews, D. F. (1975). The reporting of computation-based results in statistics. *The American Statistician*, 29 (3), pp. 122-126.

[Holland, 1962] Holland, J. H. (1962). Outline for a logical theory of adaptive systems. *Journal of the Association for Computing Machinery*, 3, pp.297-314.

[Holland, 1975] Holland, J. H. (1975). Adaptation in natural and artificial systems. The University of Michigan Press, Ann Arbor, MI.

[Hooke et Jeeves, 1961] Hooke, R., and Jeeves, T. A. (1961). Direct search solution of numerical and statistical problems. *Journal of the Association of Computing Machinery*, pp. 221-224.

[Hu, 1992] Hu, N. (1992). Tabu search method with random moves for globally optimal design. *International Journal for Numerical Engineering*, 35, pp. 1055-1070.

[Huyet et Paris, 2003] Huyet, A. L., and Paris, J. L. (2003). Configuration and analysis of a multiproduct kanban system using evolutionary optimization coupled to machine learning. *CESA'2003*, Lille.

[Hynynen, 1988] Hynynen, J. (1988). A framework for coordination in distributed production management. *PhD thesis, Helsinki University of Technology*, Finland.

[Igel et Toussaint, 2003] Igel, C., and Toussaint, M. (2003). On Classes of functions for which no free lunch results hold. *Information Processing Letters*, 86 (6), pp. 317-321.

[Ingber et Rosen, 1992] Ingber, L., and Rosen, B. (1992). Genetic algorithms and very fast simulated reannealing: A comparison. *Mathl. Comput. Modelling*, 16:87-100.

[Ingber, 1995] Ingber, L. (1995). Adaptive simulated annealing (ASA) (ftp://alumni.caltech.edu/pub/ingber/).

[Jain et Meeran, 1998] Jain, A., and Meeran, S. (1998) A state-of-the-art review of job-shop scheduling techniques. *Technical report*, Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Dundee.

[Janikow et Michalewicz, 1991] Janikow, C. Z., and Michalewicz, Z. (1991). An experimental comparison of binary and floating point representations in genetic algorithms. *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 31-36.

[Jin, 2003] Jin, Y. (2003). A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing Journal*.

[Johnson et McGeoch, 1997] Johnson, D. S., and McGeoch, L. A. (1997). The traveling salesman problem: A case study in local optimization. In Aarts, E. H. L., and Lenstra, J. K. (eds.), *Local Search in Combinatorial Optimization*, Wiley and Sons.

[Jones et Rabelo, 1998] Jones, A. and Rabelo, L. C. (1998). Survey of job shop scheduling techniques. *NISTIR, National Institute of Standards and Technology*, Gaithersburg, MD.

[Juels et Wattenberg, 1996] Juels, A., and Wattenberg, M. (1996). Stochastic hillclimbing as a baseline method for evaluating genetic algorithms. In Touretzky, D. S., Mozer, M. C. and Hasselmo, M. E. (eds.), *Advances in Neural Information Processing Systems*, 8, pp. 430-436.

[Kalagnanam et Diwekar, 1997] Kalagnanam, J. R., and Diwekar, U. M. (1997). An efficient sampling technique for Off-Line Qality Control. *Technometrics*, 39 (3), pp. 308-319.

[Kargupta, 1995] Kargupta, H. (1995). SEARCH, polynomial complexity, and the fast messy genetic algorithm. *Illigal Report No. 95008*.

[Karimi et al., 1996] Karimi, K. J., Booker, A., and Mong, A. (1996). Modeling simulation and verification of large DC power electronics systems. *Proceedings of the 27th IEEE Power Electronics Specialists Conference*, pp. 1731-1738.

[Kennedy et Eberhart, 1995] Kennedy, J., and Eberhart, R. C. (1995). Particle swarm optimization. *Proceedings of the IEEE Conference on Neural Networks*, IV, Piscataway, NJ, pp. 1942-1948.

[Kennedy, 1998] Kennedy, J. (1998). The behavior of particles! *Evol. Progr. VII*, pp. 581-587.

[Kennedy et al., 2001] Kennedy, J., Eberhart, R. C., and Shi, Y. (2001). Swarm intelligence. Morgan Kaufmann Publishers, San Francisco.

[Kita et Yamamura, 1999] Kita, H., and Yamamura, M. (1999). A functional specialization hypothesis for designing genetic algorithms. *Proc. IEEE International Conference on Systems, Man and Cybernetics*, pp. III-579-584.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220 (4598), pp. 671-680.

[Kleijnen, 1987] Kleijnen, J. P. C. (1987). Statistical tools for simulation practitioners. Marcel Dekker, NY.

Kohavi et John, 1997 Kohavi, R., and John, G. (1997). Wrappers for feature selection. *Artificial Intelligence*, 97 (1-2), December, pp. 273-324.

[Koza, 1992] Koza, J. R. (1992). Genetic Programming. MIT Press.

[Krottmaier, 1993] Krottmaier, J. (1993). Optimizing engineering designs. McGraw-Hill, London.

[Ku et Mak, 1998] Ku, K. W. C., and Mak, M. W. (1998). Empirical analysis of the factors that affect the Baldwin effect. In Eiben, A. E., Bäck, T., Schonauer, M., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature – PPSN V*, Berlin, Springer, pp. 481-490.

[Kvasnicka et Pospichal, 1997] Kvasnicka, V., and Pospichal, J. (1997). A hybrid of simplex method and simulated annealing. *Chemometrics and Intelligent LaboratorySystems*, 39, pp. 161-173.

[Laguna, 2002] Laguna, M. (2002). Scatter Search. *In Handbook of Applied Optimization*, P. M. Pardalos and M. G. C. Resende (eds.), Oxford University Press, New York, pp. 183-193.

[Laguna et Adenso-Diaz, 2002] Laguna, M., and Adenso-Diaz, B. (2002). Fine-tuning of algorithms using fractional experimental designs and local search (http://leeds.colorado.edu/faculty/laguna/articles/ finetune.pdf).

[Langley, 1994] Langley, P. (1994). Selection of relevant features in machine learning. *Proceedings of the AAAI Fall Symposium on Relevance*, pp. 1-5.

[Larrañaga et al., 1999] Larrañaga, P., Etxeberria, R., Lozano, J. A., Sierra, B., Inza, I., and Peña, J. M. (1999). A review of the cooperation between evolutionary computation and probabilistic graphical models. *Proc. of the Second Symposium on Artificial Intelligence, CIMAF 99, Adaptive Systems*, pp. 314-324.

[Lawler et al., 1993] Lawler, E. L., Lenstra, J. K., Rinnooy Kann, A. H. G., and Shmoys, D. B. (1993). Sequencing and scheduling: algorithms and complexity. In S. C. Graves, A. H. G. Rinnooy Kan, and P. H. Zipkin (Eds) *Logistics of Production and Inventory*, Amsterdam: Elsevier, pp. 445-522.

[Lawrence, 1984] Lawrence, S. (1984). Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques. *Technical report*, GSIA, Carnegie Mellon University.

[Lereno et al., 2001] Lereno, E., Morello, B., and Baptiste, P. (2001). Système d'aide au paramétrage d'un logiciel en ordonnancement. *3ème Conférence Francophone de modélisation et simulation MOSIM'01*, pp. 363-369.

Liu, H., and Yu, L. (2003). Feature selection for data mining. *Survey draft, to be published*, http://www.public.asu.edu/~huanliu/sur-fs02.ps.

[Lourenço et al., 2002] Lourenço, H. R., Martin, O., and Stützle, T. (2002). Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*. Kluwer Academic Publishers, Norwell, MA, pp. 321-353.

[Maron et Moore, 1994] Maron, O., and Moore, A. (1994). Hoeffding races: Accelerating model selection search for classification and function approximating. *In Advances in Neural Information Processing Systems*, pp. 59-66.

[Martin et al., 1991] Martin, O., Otto, S. W., and Felten, E. W. (1991). Large-step markov chains for the travelling salesman problem. *Complex Systems*, 5 (3), pp. 299-326.

[McIlhagga, 1997] McIlhagga, M. (1997). Solving generic scheduling problems with a distributed genetic algorithm. *In proceedings of the AISB Workshop on Evolutionary Computing*, pp. 85-90.

[McGeoch et Moret, 1999] McGeoch, C. C., and Moret, B. M. E. (1999). How to present a paper on experimental work with algorithms (http://www.cs.unm.edu/~moret/howto.ps).

[Mckay et al., 1979] McKay, M. D., Beckman, R. J., and Conover, W. J. (1979). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21 (2), pp. 239-245.

[Merz, 2000] Merz, P. (2000). Memetic algorithms for combinatorial optimization problems: Fitness landscapes and effective search strategies. *Phd thesis*, Department of Electrical Engineering and Computer Science, University of Siegen, Germany.

[Michalewicz, 1994] Michalewicz, Z. (1994). Genetic Algorithms + Data Structures = Evolution Program. Springer, third edition.

[Michalewicz, 1999] Michalewicz, Z. (1999). Genetic algorithms + Data structures = evolution programs. Springer.

[Miettinen, 1999] Miettinen, K. M. (1999) Nonlinear multiobjective optimization. Kluwer academic publisher editions.

[Milin, 1987] Milin, A.M. (1987). Amélioration des solutions d'ordonnancement pour le pilotage d'atelier. *Thèse de doctorat*, Université de Bordeaux I, France.

[Miller, 1990] Miller, A. (1990). Subset selection in regression. London: Chapman and Hall.

[Mladenovic et Hansen, 1997] Mladenovic, N., and Hansen, P. (1997). Variable neighborhood search. *In Computers in Operations Research*, 24 (11), pp. 1097-1100.

Molina, L. C., Belanche, L., and Nebot, À. (2002). Feature selection algorithms: a survey and experimental evaluation. *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002)*, 9-12 December 2002, Maebashi City, Japan, IEEE Computer Society, pp. 306-313.

[Mongeau et al., 2000] Mongeau, M., Karsenty, H., Rouzé, V., and Hiriart-Urruty, J.-B. (2000). Comparison of public-domain software for black box global optimization. *Optimization Methods and Software* 13, pp. 203-226.

[Montana, 2001] Montana, D. J. (2001). A reconfigurable optimizing scheduler. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'2001*, San Francisco.

[Montazeri et Wassenhove, 1990] Montazeri M., and Van Wassenhove L.N. (1990). Analysis of dispatching rules for an FMS*, International Journal of Production Research*, 28 (4), pp 785-802.

[Montgomery, 2001] Montgomery, D. C. (2001). Design and analysis of experiments. Fifth Edition, John Wiley & Sons, New York.

[Moore et Lee, 1994] Moore, A., and Lee, M. (1994). Efficient algorithms for minimizing cross validation error. *In ICML-9*, pp. 190-198.

[Morris et al., 1998] Morris, G. M., Goodsell, D. S., Halliday, R. S., Huey, R., Hart, W. E., Belew, R. K. and Olson, A. J. (1998). Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function. *Journal of Computational Chemistry*, 19 (14), pp. 1639-1662.

[Moscato, 1989] Moscato, P. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *C3P Report 826, Caltech Concurrent Computation Program*, Caltech, California, USA.

[Moscato, 1993] Moscato, P. (1993). An introduction to population approaches for optimization and hierarchical objective functions: A discussion on the role of tabu search. *In Annals of Operations Research*, 41, pp. 85-121.

[Muhl et al., 2003] Muhl, E., Charpentier, P., and Chaxel, F. (2003). Automotive manufacturing plant physical flows optimization; some experiments and issues. *To be published in Engineering Applications of Artificial Intelligence*.

[Mühlenbein et Schlierkamp, 1993] Mühlenbein, H., and Schlierkamp-Voosen, D. (1993). Predictive Models for the Breeder Genetic Algorithm I, Continuous parameter optimization. *Evolutionary Computation*, 1 (1), pp. 25-49.

[Mühlenbein et Schlierkamp, 1994] Mühlenbein, H., and Schlierkamp-Voosen, D. (1994). The science of breeding and its application to the breeder genetic algorithm. *Evolutionary computation*, 1, pp. 335-360.

[Mühlenbein et Mahnig, 1999] Mühlenbein, H., and Mahnig, T. (1999). FDA – A scalable evolutionary algorithm for the optimization of additively

decomposed functions. *Evolutionary Computation*, 7 (4), pp. 353-376.

[Murray, 1972] Murray, W. (1972). Numerical methods for unconstrained optimization, Academic Press.

[Nakano et Yamada, 1991] Nakano, R. and Yamada, T. (1991). Conventional genetic algorithms for job shop problems. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, Morgan Kaufmann, pp. 474-479.

[Nasereddin et Mollaghasemi, 1999] Nasereddin, M. and Mollaghasemi, M. (1999). The development of a methodology for the use of neural networks and simulation modeling in system design. *Proccedings of the 1999 Winter Simulation Conference*, P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans eds., pp. 537-542.

[Nelder et Mead, 1965] Nelder, J. A., and Mead, R. (1965). A simplex method for function minimization. *Computer Journal*, 7, pp. 308-313.

[Newman, 1988] Newman, P. A. (1988). Scheduling in CIM systems. In A. Kusiak (Ed.) Artificial Intelligence in Industry: Implications for CIM, New York: Springer-Verlag, pp. 361-402.

[Norenkov, 1994] Norenkov, I. P. (1994). Scheduling and allocation for simulation and synthesis of CAD system hardware. *Proc. EWITD 94, East-West internat. Conf.*, Moscow, ICSTI, pp. 20-24.

[Norenkov et Goodman, 1997] Norenkov I. P., and Goodman E. (1997). Solving Scheduling Problems via Evolutionary Methods for Rule Sequence Optimization. *Second World Conference on Soft Computing (WSC2)*.

[Ólafsson et Kim, 2002] Ólafsson, S., and Kim, J., (2002). Simulation optimization. *Proceedings of the 2002 Winter Simulation Conference*, E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds, pp. 79-84.

[Ono et Kobayashi, 1997] Ono, I., and Kobayashi, S. (1997). A real-coded genetic algorithm for function optimization using unimodal normal distribution crossover. *Proc. Of the Seventh International Conference on Genetic Algorithms*, pp. 246-253.

[Ono et al. 1999] Ono, I., Kita, H., and Kobayashi, S. (1999). A robust real-coded genetic algorithm using unimodal normal distribution crossover augmented by uniform crossover: effects of self-adaptation of crossover probabilities. *Proc. Of the Genetic and Evolutionary Computation Conference (GECCO-99)*, Morgan Kaufmann, pp. 496-503.

[Panwalker et Iskander, 1977] Panwalker, S. S., and Iskander, W. (1977). A survey of scheduling rules. Operations Research, 25 (1), pp. 45-61.

[Papadimitriou et Steiglitz, 1982] Papadimitriou, C. H., and Steiglitz, K. (1982). Combinatorial optimization: Algorithms and complexity. Prentice Hall, New Jersey.

[Paris et Pierreval, 2001] Paris, J-L., and Pierreval, H. (2001). A distributed evolutionary simulation optimization approach for the configuration of multiproduct kanban systems. *Int. J. Computer Integrated Manufacturing*, 14 (5), pp. 421-430.

[Pelikan et al., 2000] Pelikan, M., Goldberg, D. E., and Lobo, F. (2000). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, Kluwer.

[Pierreval et Mebarki, 1997] Pierreval H., and Mebarki, N. (1997). Dynamic selection of dispatching rules for manufacturing system scheduling, *International Journal of Production Research*, vol. 35, n°6, pp 1575-1591.

[Pierreval et Paris, 2000] Pierreval, H., and Paris, J. L. (2000). Distributed Evolutionary Algorithms for Simulation Optimization. *IEEE Transactions on Systems, Man and Cybernetics*, Part A : Systems and Humans, 30 (1), January, pp 15-24.

[Pinedo, 1995] Pinedo, M. (1995). Scheduling: Theory, Algorithms and Systems. Prentice Hall.

[Pomerol et Barba-Romero, 1993] Pomerol, J. C., and Barba-Romero, S. (1993). Choix multicritère dans l'entreprise : principe et pratique, ed. Hermes.

[Powell, 1964] Powell, M. J. D. (1964). An efficient method for finding minimum of a function of several variables without calculating derivatives. *Computer Journal*, 7, pp. 155-162.

[Powell, 1987] Powell, M. J. D. (1987). Radial basis functions for multivariable interpolation. A review. In Mason, J. C. and Cox, M. G., editors, Algorithms for Approximation of Functions and Data, Oxford University Press, Oxford, pp. 143-167.

[Qi et Palmieri 1994a] Qi, A., and Palmieri, F. (1994a). Theoretical analysis of evolutionary algorithms with an infinite population size in continuous space, Part I: Basic properties of selection and mutation. *IEEE Trans. On Neural Networks*, 5 (1), pp. 102-119.

[Qi et Palmieri 1994b] Qi, A., and Palmieri, F. (1994b). Theoretical analysis of evolutionary algorithms with an infinite population size in continuous space, Part II: Analysis of the diversification role of crossover. *IEEE Trans. On Neural networks*, 5 (1), pp. 120-129.

[Quinlan, 1993] Quinlan, J. R. (1993). C4.5 : Programs for Machine Learning, Morgan kaufmann.

[Rechenberg, 1965] Rechenberg, I. (1965). Cybernetic solution path of an experimental problem. *Royal Aircraft Establishment*, Library translation No. 1122, Farnborough Hants., UK, August.

[Rechenberg, 1973] Rechenberg, I. (1973). Evolutionsstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution. Fromman-Holzboog Verlag, Stuttgart.

[Rechenberg, 1994] Rechenberg, I. (1994). Evolutionsstrategie '94. Stuttgart, Friedrich Frommann Verlag.

[Resende et Ribeiro, 2002] Resende, M. G. C. and Ribeiro, C. C. (2002). Greedy randomized adaptive search procedures. In Glover, F., Kochenberger, G., eds., *State of the Art Handbook of Metaheuristics*, Kluwer, pp. 219-249.

[Rinnooy Kan, 1976] Rinnooy Kan, A. H. G. (1976). Machine scheduling problems: Classification, complexity, and computations, The Hague, Holland: Martinus Nijhoff.

[Rinnooy Kan et Timmer, 1989] Rinnooy Kan A. H. G., and Timmer, G. T. (1989). Global Optimization. In G. L. Nemhauser, A. H. G. Rinnooy Kan and M. J. Todd, eds., *Handbooks in Operations Research and Management Science. Vol. I: Optimization*. North-Holland, Amsterdam, pp. 631-662.

[Rodammer et White, 1988] Rodammer, F. A. and White, K. P. (1988). A recent survey of production scheduling. *IEEE Transactions on Systems, Man and Cybernetics*, 18 (6), pp. 841-851.

[Rogers, 2002] Rogers, P. (2002). Optimum-seeking simulation in the design and control of manufacturing systems: Experience with OptQuest for Arena. *Proceedings of the 2002 Winter Simulation Conference*, E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds., pp. 1142-1150.

[Roy et Bouyssou, 1993] Roy, B. et Bouyssou, D. (1993). Aide Multicritère à la Décision : Méthodes et Cas, Economica.

[Rudlof et Köppen, 1996] Rudlof, S., and Köppen, M. (1996). Stochastic hill climbing with learning by vectors of normal distributions. *The first online workshop on soft computing*, (http://www.uchikawa.nuie.nagoyau.ac.jp/wsc1/papers/p077.ht ml).

[Sabuncuoglu, 1998] Sabuncuoglu, (1998). A study of scheduling rules of flexible manufacturing systems : a simulation approach. *International Journal of Production Research*, 36 (2), pp. 527-546.

[Sacks et al., 1989] Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn H. P. (1989). Design and analysis of computer experiments. *Statistical Science*, 4, pp. 409-435.

[Sage, 2003] Sage, A. (2003). Observation-driven configuration of complex software systems. *Phd thesis*, School of Computer Science, University of St Andrews, Scotland.

[Sauer et al., 1997] Sauer, W., Weigert, G., and Hampel, D. (1997). An open optimization system for controlling of manufacturing processes. *In 7th International Conference FAIM '97, Flexible Automation & Intelligent Manufacturing*, Middlesbrough, England, June 1997, p. 261-268.

[Schaffer et al., 1989] Schaffer, J. D., Caruana, R. A., Eshelman, L. J., and Das, R., (1989). A study of control parameters affecting online performance of genetic algorithms for function optimization. In *Proceedings of the Third International Conference on Genetic Algorithms*, George Mason University, pp. 51-60.

[Schoenauer et Michalewicz, 1997] Schoenauer, M., and Michalewicz, Z. (1997). Evolutionary Computation: An Introduction. *Control and Cybernetics, Special Issue on Evolutionary Computation*, 26 (3), pp. 307-338.

[Schwefel, 1965] Schwefel, H.-P. (1965). Kybernetische Evolution als Strategie der experimentellen Forshung in der Strömungstechnik. *Diplomarbeit, Technische Universität Berlin.*

[Schwefel, 1977] Schwefel, H.-P. (1977). Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie. *In Interdisciplinary Systems Research*, 26, Birkhäuser, Basel.

[Schwefel, 1981] Schwefel, H.-P. (1981). Numerical Optimization of Computer Models. John Wiley & Sons, New-York.

[Schwefel, 1995] Schwefel, H.-P. (1995). Evolution and optimum seeking. *Sixth-Generation Computer Technology Series*. Wiley, New York.

[Sebag et Ducoulombie, 1998] Sebag, M., and Ducoulombier, A. (1998). Extending population-based incremental learning to continuous search spaces. *Parallel Problem Solving from Nature – PPSN V*, Springer Verlag, pp. 418-427.

[Serafini, 1998] Serafini, D. B. (1998). A framework for managing models in nonlinear optimization of computationally expensive functions. *Phd thesis*, Rice university, Houston, Texas.

[Servet et al., 1997] Servet, I., Trave-Massuyes, L., and Stern, D. (1997). Telephone network traffic overloading diagnosis and evolutionary computation techniques. *Proc. Of the Third European Conf. on Artificial Evolution*, pp. 137-144.

[Shang and Tadikamalla, 1993] Shang, J. S., and Tadikamalla, P. R. (1993). Output maximization of a CIM system: simulation and statistical approach. *International Journal of Production Research*, 31 (1), pp. 19-41.

[Shang, 1995] Shang, J. S. (1995). Robust design and optimization of material handling in an FMS. *International Journal of Production Research*, 33 (9), pp. 2437-2454.

[Shang and Tadikamalla, 1998] Shang, J. S., and Tadikamalla, P. R. (1998). Multi-criteria design and control of cellular manufacturing system through simulation and optimization. *International Journal of Production Research*, 36 (6), pp. 1515-1528.

[Sheskin, 1997] Sheskin, D. J. (1997). Handbook of parametric and nonparametric statistical procedures. CRC Press, Boca Raton, Florida.

[Siarry et Berthiau, 1997] Siarry, P., and Berthiau, G. (1997). Fitting of tabu search to optimize functions of continuous variables. *International Journal for Numerical Methods in Engineering*, 40, pp. 2449-2457.

[Siarry et al., 1997] Siarry, P., Berthiau, G., Durbin, F., and Haussy, J. (1997). Enhanced simulated annealing for globally minimizing functions of many continuous variable. *ACM Trans. Mathematical Software*, 23 (2), pp. 209-228.

[Sim et al., 1994] Sim, S. K., Yeo, K. T., and Lee, W. H. (1994). An expert neural network system for dynamic jobshop scheduling. *International Journal of Production Research* 32 (8), pp. 1759-1773.

[Simpson et al., 1997] Simpson, T. W., Peplinski, J., Koch, P. N., and Allen, J. K. (1997). On the use of statistics in design and the implications for deterministic computer experiments. *Design Theory and Methodology – DTM'97*, Sacramento, CA, ASME, Paper No. DETC97/DTM-3881.

[Simpson et al., 1998] Simpson, T. W., Mauery, T. M., Korte, J. J., and Mistree, F. (1998). Comparison of response surface and kriging models for multidisciplinary design optimization. *7th Symposium on Multidisciplinary Analysis and Optimization*, St. Louis, MO, AIAA-98-4755.

[Smith et al., 1990] Smith, S.F., Ow, P.S., Potvin, J.Y., Muscettola, N., and Matthys, D. (1990). OPIS: An Opportunistic Factory Scheduling System. Proceedings of the Third International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE-90), Knoxville, USA, May.

[Smith, 1992] Smith, S. F. (1992). Knowledge-based production management: approaches, results, and prospects. *Production Planning and Control*, 3 (4), pp. 350-380.

[Smith, 1993] Smith, M. (1993). Neural networks for statistical modeling. Von Nostrand Reinhold, New York.

[Smith et Becker, 1997] Smith, S., and Becker, M. (1997). An ontology for constructing scheduling systems. *In working notes of 1997 AAAI Symposium on Ontological Egineering*.

[Sobieszczanski et Haftka, 1997] Sobieszczanski-Sobieski, J., and Haftka, R. T. (1997). Multidisciplinary aerospacedesign optimization: Survey of recent developments. *Structural optimization, 14*, pp. 1-23.

[Spendley et al., 1962] Splendley, W., Hext, G. R., and Himsworth, F. R. (1962). Sequential application of simplex designs in optimization and evolutionary operation. *Technometrics, 4*, pp. 441-461.

[Steffen, 1986] Steffen, M.S. (1986). A survey of artificial intelligence-based scheduling systems, *Proceedings of the IEE Fall Industrial Conference*, Boston, MA, pp. 395-405.

[Steuer, 1986] Steuer, R. (1986). Multiple criteria optimization: theory, computation and application. New York, John Wiley & Sons, Inc.

[Stützle, 1999a] Stützle, T. (1999). Iterated local search for the quadratic assignment problem. *Technical report aida-99-03, FG Intellektik*, TU Darmstadt.

[Stützle, 1999b] Stützle, T. (1999). Local search algorithms for combinatorial problems – analysis, algorithms and new applications. DISKI – *Dissertationen zur Künstliken Intelligenz,* infix.

[Subramaniam et al., 2000] Subramaniam, V., Lee, G. K., Hong, G. S., Wong, Y. S., and Ramesh, T. (2000). Dynamic selection of dispatching rules for job shop scheduling. *Production planning & control*, 11 (1), pp. 73-81.

[Subramaniam, 1995] Subramniam, V. (1995). Scheduling of manufacturing systems based on extreme value theory and genetic algorithms. *Phd thesis*, Massachusetts Institute of Technology, USA.

[Surry et Radcliffe, 1996] Surry, P. D., and Radcliffe, N. (1996). Real representations. *Foundations of Genetic Algorithms 4*, Morgan Kaufman Publishers, San Francisco, pp. 343-363.

[Syrjakow et Szczerbicka, 1994] Syrjakow, M., and Szczerbicka, H. (1994). Optimization of Simulation Models with REMO. *Proceedings of the European Simulation Multiconference ESM'94*, Barcelona, Spain, June 1-3, pp. 274-281.

[Szu et Hartley, 1987] Szu, H. H., and Hartley, R. L. (1987). Noneconomic optimization by fast simulated annealing. *Proceedings of the IEEE*, 75, pp. 1538-1540.

[Taillard, 1991] Taillard, E. D. (1991). Robust tabu search for the quadratic assignment problem. *In Parallel Computing*, 17, pp. 443-455.

[Taillard, 1995] Taillard, E. D. (1995). Comparison of iterative searches for quadratic assignment problem. *In Location Science*, 3, pp. 87-105.

[Taillard, 2001] Taillard, E. D. (2001). Comparison of non-deterministic iterative methods. *INA working paper, EIVD,* Yverdon, Switzerland, March.

[Talbi, 2000] Talbi, E. G. (2000). Metaheuristics for multiojective combinatorial optimization: state of the art. *Technical report*, LIFL, University of Lille, France.

[Talbi et al., 2002] Talbi, D., Geneste, L., Grabot, B., Prévitali, R., and Hostachy, P. (2002). Optimal Set-up of an Industrial Scheduling Software. *APMS 2002 Collaborative Systems for Production Management*, Eindhoven, Pays-Bas, September 8-13.

[Talbi et al., 2003] Talbi, E. D., Geneste, L., and Grabot, B. (2003). Meta-heuristics for optimal set-up of an industrial scheduling software. *CESA'2003, 9-11 juillet 2003*, Lille, France.

[Talbi et al., 2004a] Talbi, E. D., Grabot, B. et Geneste, L. (2004a). Algorithmes évolutifs pour le paramétrage d'un logiciel d'ordonnancement. *MOSIM'04*, Nantes, France.

[Talbi et al., 2004b] Talbi, E. D., Geneste, L., Grabot, B., Previtali, R., and Hostachi, P. (2004b). Application of optimization techniques to parameter set-up of industrial scheduling software. *Computers In Industry* 55, pp. 105-124.

[T'Kindt et Billaut, 2002] T'Kindt, V. and Billaut, J.-C. (2002). Multicriteria scheduling: theory, models and algorithms. Springer-Verlag, Heidelberg.

[Toal et al., 1994] Toal D., Coffey, T., and Smith, P. (1994). Expert systems and simulation in scheduling, *Proc. IMC11*, Belfast.

[Torczon, 1989] Torczon, V. (1989). Multi-Directional Search: A direct search algorithm for parallel machines. *Phd thesis*, Rice University.

[Torczon, 1997] Torczon, V. (1997). On the convergence of pattern search algorithms. *SIAM J. Optimization*, 7 (1), pp. 1-25.

[Trosset et Torczon, 1997] Trosset, M. W., and Torczon, V. (1997). Numerical optimization using computer experiments. *NASA CR-201724 ICASE Report No. 97-38*, Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center Hampton, VA 23681-0001.

[Tsallis et Stariolo, 1996] Tsallis, C., and Stariolo, D. A. (1996). Generalized simulated annealing. *Physica A*, 233 (1-2), pp. 395-406.

[Unal et al., 1996] Unal, R., Lepsch, R. A., Engelund, W., and Stanley, D. O. (1996). Approximation model building and multidisciplinary design optimization using response surface methods. *AIAA Paper 96-4044, 6th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Bellevue, Washington, September.

[Van Breedam, 1995] Van Breedam, A. (1995). Improvement heuristics for the vehicle routing problem based on simulated annealing. *European Journal of Operational Research*, 86, pp. 480-490.

[Van der Pluym, 1990] Van Der Pluym, B. (1990). Knowledge-based decision making for job-shop scheduling. *International Journal of Integrated Manufacturing*, 6 (3), pp. 354-363.

[Venter et al., 1996] Venter, G., Haftka, R. T., and Starnes, J. H. Jr. (1996). Construction of response surfaces for design optimization applications. *AIAA Paper 96-4040, 6th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Bellevue, Washington, September.

[Vetterling et al., 1994] Vetterling, W. T., Press, W. H., Teukolsky, S. A., and Flannery, B. P. (1994). Numerical recipes in Fortran: the art of scientific computing. Cambridge University, New York.

[Voß, 2001] Voß, S. (2001). Metaheuristics: The state of the art. In A. Nareyek, editor, Local Search for Planning and Scheduling, volume 2148 of *Lecture Notes in Computer Science*, pp. 1–23, Springer, Heidelberg, Germany.

[Voudouris et Tsang, 1995] Voudouris, C., and Tsang, E. (1995). Guided Local search. *Technical Report, CSM-217*, Department of Computer Science, University of Essex.

[Wegener, 2000] Wegener, I. (2000). On the design and analysis of evolutionary algorithms. *Proceedings of the Workshop on Algorithm Engineering as a New Paradigm*, pp. 36-47.

[Weigert et al., 1998] Weigert, G., Hampel, D., and Sauer, W. (1998). Scheduling of FMS - Application of an open optimization system. *8th International Conference FAIM'98, Flexible Automation & Intelligent Manufacturing*, Portland/Oregon, USA, proceedings pp. 771-780.

[Weigert et al., 2000] Weigert, G., Werner, S., Hampel, D., Heinrich, H., and Sauer, W. (2000). Multi objective decision making - Solutions for the optimization of manufacturing processes. *10th International Conference FAIM'2000*, Flexible Automation & Intelligent Manufacturing Maryland, USA, June 2000, Proceedings, pp. 487-496.

[Whitley, 1989] Whitley, D. (1989). The GENITOR algorithm and selection pressure: Why rank-based allocation of reproduction trials is best. *In Proceedings of the third International conference on Genetic Algorithms*, San Mateo, California, USA, Morgan Kaufmann publishers, pp. 116-121.

[Whitley, 2000] Whitley, L. D. (2000). Functions as permutations: Implications for No Free Lunch. Walsh Analysis and Statistics, in Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J. J. and Schwefel, H.-P. (eds.), *Proc. Of the Sixth International Conference on Parallel Problem Solving from Nature (PPSN VI)*, Springer Verlag, Berlin, pp. 169-178.

[Wolpert et Macready, 1995] Wolpert, D. H., and Macready, W. G. (1995). No free lunch theorems for search. *Santa Fe Institute Technical Report SFI-TR-05-010*, Santa Fe Institute, Santa Fe, NM.

[Wolpert et Macready, 1997] Wolpert, D. H., and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1 (1), pp. 67-82.

[Wright, 1991] Wright, A. H. (1991). Genetic algorithms for real parameter optimization. *Foundations of genetic Algorithms, Morgan Kaufman Publishers*, San Mateo, pp. 205-218.

[Wu, 1987] Wu, D. (1987). An Expert Systems Approach for the Control and Scheduling of Flexible Manufacturing Systems. *Ph.D. Dissertation*, Pennsylvania State University.

[Xu et al., 1998] Xu, J., Steve, Y., and Glover, F. (1998). Fine-tuning a tabu search algorithm with statistical tests. *International Transactions in Operational Research*, 5 (3), pp. 233-244.

[Zadeh, 1965] Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, 8, pp. 338-353.

[Zighed et Rakotomalala, 2000] Zighed, D. A., and Rakotomalala, R. (2000). Graphes d'induction. Edition Hermes.

Résumé

L'utilisation d'un logiciel d'ordonnancement industriel fait intervenir une multitude de paramètres dont le réglage influence fortement la qualité des résultats. A l'heure actuelle, ce réglage est effectué de façon manuelle, après un travail souvent fastidieux au cours de l'installation initiale du logiciel. De plus, une fois spécifiées, les valeurs de ces paramètres sont rarement remises en cause par les utilisateurs, du fait de leur manque d'expérience et du nombre important de paramètres à ajuster. L'idée que nous développons ici consiste à utiliser des métaheuristiques pour automatiser cette tâche. Deux problèmes seront abordés : la sélection des paramètres pertinents et leur réglage en fonction des exigences de l'utilisateur. Nous proposons de résoudre ces deux problèmes de façon simultanée, en introduisant des stratégies de sélection au sein des métaheuristiques. Cette approche est appliquée au logiciel d'ordonnancement Ortems® et validée sur plusieurs cas industriels.

Mots-clés : optimisation en boîte noire, ordonnancement, métaheuristiques, sélection de paramètres.

Abstract

PARAMETER SELECTION AND TUNING FOR OPTIMIZATION OF INDUSTRIAL SCHEDULING SOFTWARES

The use of scheduling software requires to set-up a number of parameters that have a direct influence on the schedule quality. Nowadays, this set-up is obtained manually after an extensive effort during initial software installation. Moreover, this set-up is rarely called into question by users, due to their lack of experience and to the high number of parameters involved. It is suggested in this thesis the use of metaheuristics to automate this task. Two problems are considered: selection of relevant parameters and their tuning according to user requirements. We suggest here an approach to solve these problems simultaneously, based on the combination of metaheuristics with some parameter selection strategies. An implementation framework has been developed and tested on an industrial scheduler, named Ortems®. The first results of the use of this framework on real industrial databases are described and commented.

Keywords: black box optimization, scheduling, metaheuristics, parameter selection.